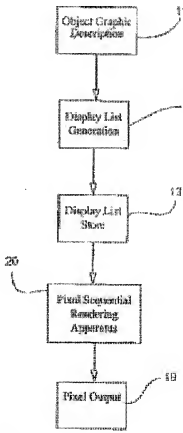
METHOD AND DEVICE FOR PROCESSING GRAPHIC OBJECT FOR HIGH-SPEED RASTER FORM RENDERING

Publication number: JP2000149035 (A)
Publication date: 2000-05-30
Inventor(s): TIMOTHY MERRICK LONG; CHRISTOPHER FRASER; KEVIN MOORE
Applicant(s): CANON KK
Classification:
- international: ***G06T11/40; G09G5/42;*** **G06T11/40; G09G5/42;** (IPC1-7): G06T11/40
- European: G06T11/40; G09G5/42
Application number: JP19990255123 19990909
Priority number(s): AU1998PP05854 19980911; AU1998PP05858 19980911; AU1998PP05859 19980911; AU1998PP05862 19980911; AU1999PP09234 19990316; AU1999PQ00049 19990429

Also published as:
US6483519 (B1)
JP2006338692 (A)

Abstract of **JP 2000149035 (A)**

PROBLEM TO BE SOLVED: To decide crossing order among sides of a graphic object and to decide on the crossing value of the sides regarding each side for a following scan line by evaluating side record corresponding to the present scan line. SOLUTION: Object graphic description 11 is generated by a host processor or guided from a system ROM and is used for describing parameters of the graphic object. The object graphic description 11 is interpreted by a pixel sequential rendering device 20 for rendering a pixel base image from which it is generated. For example, an object with sides is incorporated in several formats, including an orthogonal side format in which a two-dimensional object, is defined by a straight side (a simple vector) to cross from one point to another on a display or plural sides, including orthogonal lines in the object graphic description 11.



Family list
12 application(s) for: **JP2000149035 (A)**

1 **Fast renering techniques for rasterised graphic object based images**
Inventor: LONG TIMOTHY MERRICK ; LIE TJOAN (+2) **Applicant:** CANON KK
EC: **IPC:** ***G06T11/00;*** **G06T11/00;** (IPC1-7): G06T11/00
Publication info: **AU743218 (B2)** — 2002-01-24

2 **Processing graphic objects for fast rasterised rendering**
Inventor: LONG TIMOTHY MERRICK ; FRASER CHRISTOPHER (+1) **Applicant:** CANON KK
EC: **IPC:** ***G06T1/60;*** **G06T1/60;** (IPC1-7): G06T1/60
Publication info: **AU744091 (B2)** — 2002-02-14

3 **Compositing objects with opacity for fast rasterised rendering**
Inventor: LONG TIMOTHY MERRICK **Applicant:** CANON KK
EC: **IPC:** ***G06T1/60;*** **G06T1/60;** (IPC1-7): G06T1/60
Publication info: **AU779154 (B2)** — 2005-01-06

4 **Compositing objects with opacity for fast rasterised rendering**
Inventor: LONG TIMOTHY MERRICK **Applicant:** CANON KK
EC: **IPC:** ***G06T1/60;*** **G06T1/60;** (IPC1-7): G06T1/60
Publication info: **AU4064502 (A)** — 2002-06-27

5 **Processing graphic objects for fast rasterised rendering**
Inventor: LONG TIMOTHY MERRICK ; MOORE KEVIN JOHN (+1) **Applicant:** CANON KK
EC: **IPC:** ***G06T1/60;*** **G06T1/60;** (IPC1-7): G06T1/60
Publication info: **AU4750899 (A)** — 2000-03-16

6 **Fast rendering techniques for rasterised graphic object based images**
Inventor: LONG TIMOTHY MERRICK ; MOORE KEVIN JOHN (+2) **Applicant:** CANON KK
EC: **IPC:** ***G06T11/00;*** **G06T11/00;** (IPC1-7): G06T11/00
Publication info: **AU4750999 (A)** — 2000-03-16

7 **FAST RENDERING METHOD FOR IMAGE USING RASTER TYPE GRAPHIC OBJECT**
Inventor: TIMOTHY MERRICK LONG ; KOK CHANG LEE (+2) **Applicant:** CANON KK
EC: G06T11/40 **IPC:** ***G09G5/377; G06F17/00; G06T11/20;*** (+8)
Publication info: **JP2000137825 (A)** — 2000-05-16

8 **METHOD AND DEVICE FOR PROCESSING GRAPHIC OBJECT FOR HIGH- SPEED RASTER FORM RENDERING**
Inventor: TIMOTHY MERRICK LONG ; CHRISTOPHER FRASER (+1) **Applicant:** CANON KK
EC: G06T11/40; G09G5/42 **IPC:** ***G06T11/40; G09G5/42;*** **G06T11/40;** (+2)
Publication info: **JP2000149035 (A)** — 2000-05-30

9 **IMAGE PROCESSOR AND METHOD**
Inventor: TIMOTHY MERRICK LONG ; **Applicant:** CANON KK

CHRISTOPHER FRASER (+1)
EC: G06T11/40; G09G5/42 **IPC:** ***G06T11/00; G06T11/40; G09G5/42;*** (+3)
Publication info: **JP2006338692 (A)** — 2006-12-14

10 **Processing graphic objects for fast rasterised rendering**
Inventor: LONG TIMOTHY MERRICK [AU] ; FRASER CHRISTOPHER [GB] (+1) **Applicant:** CANON KK [JP]
EC: G06T11/40; G09G5/42 **IPC:** ***G06T11/40; G09G5/42;*** **G06T11/40;** (+2)
Publication info: **US6483519 (B1)** — 2002-11-19

11 **Fast rendering techniques for rasterised graphic object based images**
Inventor: LONG TIMOTHY MERRICK [AU] ; LIE KOK TJOAN [AU] (+2) **Applicant:** CANON KK [JP]
EC: G06T11/40 **IPC:** ***G09G5/377; G06F17/00; G06T11/20;*** (+6)
Publication info: **US6828985 (B1)** — 2004-12-07

12 **Processing graphic objects for fast rasterised rendering**
Inventor: LONG TIMOTHY MERRICK [AU] ; FRASER CHRISTOPHER [GB] (+1) **Applicant:** LONG TIMOTHY MERRICK, ; FRASER CHRISTOPHER, (+2)
EC: G06T11/40; G09G5/42 **IPC:** ***G06T11/40; G09G5/42;*** **G06T11/40;** (+2)
Publication info: **US2003016221 (A1)** — 2003-01-23
US7046253 (B2) — 2006-05-16

Data supplied from the ***esp@cenet*** database — Worldwide

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（A）　(11)特許出願公開番号
特開2000－149035
（P2000－149035A）
(43)公開日　平成12年５月30日（2000.5.30）

(51)Int.Cl.⁷	識別記号	ＦＩ	テーマコード（参考）
Ｇ０６Ｔ　11/40		Ｇ０６Ｆ　15/72　４００	

審査請求　未請求　請求項の数75　ＯＬ　外国語出願　（全177頁）

(21)出願番号　特願平11－255123
(22)出願日　平成11年９月９日(1999.9.9)
(31)優先権主張番号　ＰＰ５８６４
(32)優先日　平成10年９月11日(1998.9.11)
(33)優先権主張国　オーストラリア（ＡＵ）
(31)優先権主張番号　ＰＰ５８５８
(32)優先日　平成10年９月11日(1998.9.11)
(33)優先権主張国　オーストラリア（ＡＵ）
(31)優先権主張番号　ＰＰ５８５９
(32)優先日　平成10年９月11日(1998.9.11)
(33)優先権主張国　オーストラリア（ＡＵ）

(71)出願人　000001007
キヤノン株式会社
東京都大田区下丸子３丁目30番２号
(72)発明者　ティモシー　メリック　ロング
オーストラリア国　2113　ニューサウスウェールズ州、　ノース　ライド、　トーマス　ホルト　ドライブ　１、　キヤノン　インフォメーション　システムズ　リサーチ　オーストラリア　プロプライエタリー　リミテッド　内
(74)代理人　100076428
弁理士　大塚　康徳　（外２名）

最終頁に続く

(54)【発明の名称】　高速ラスタ形式レンダリングのためのグラフィックオブジェクト処理方法および装置

(57)【要約】　（修正有）
【課題】　複数の画素位置とを有するラスタ画素画像にレンダリングするための方法、装置およびコンピュータ可読媒体を提供する。
【解決手段】　所定の順序で、そのスキャン・ラインと交差するオブジェクトの辺の交差座標の決定は、複数のバッファを使用して辺レコードを処理し、これによって、順序への辺交差の効率的なソートを可能にすることによって達成される。辺交差の隣接する対のそれぞれについて、対応するオブジェクトに関連する情報を検査して、辺交差の対応する対の間の画素位置のスパンに関するアクティブ・オブジェクトの組を決定する。画素位置のスパンのそれぞれについて、アクティブ・オブジェクトの対応する組を使用して、スパン内の位置のそれぞれの値を決定する。

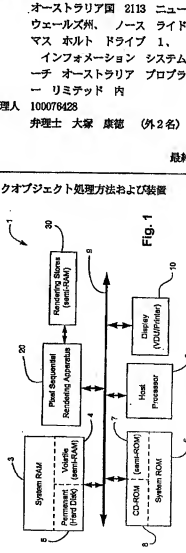


Fig. 1

【特許請求の範囲】

【請求項１】　ラスター画素イメージを形成するべく、グラフィックオブジェクトを処理する方法であって、該処理は、ラスタライズされた表示順における現在のスキャンラインのための対応する辺レコードを評価することにより、前記グラフィックオブジェクトの辺間の交差順序を判断し、後続のスキャンラインのための各辺に関する辺交差値を決定するものであって、該処理は前記辺レコードの処理の間に、
限られた数の処理された辺レコードを順序付けされていない第１バッファに保持し、順序付け可能な処理済の辺レコードが前記第１バッファに加えられのに応じて、順序付け可能な前記辺レコードを漸進的に順次に第２バッファへ送るステップと、
順序付けできない処理済の辺を、第３バッファにおいて並べるために該第３バッファへ送るステップと、
前記第２及び第３バッファからの辺レコードを選択的に処理し、後続のスキャンラインに対する並べられた交差を決定するステップとを備えることを特徴とする方法。

【請求項２】　前記第３バッファにおける前記辺レコードの順序付けは、前記第３バッファへ前記辺を追加する際になされることを特徴とする請求項１に記載の方法。

【請求項３】　前記辺レコードは、前記第３バッファへ挿入され、ソートされることを特徴とする請求項２に記載の方法。

【請求項４】　前記第３バッファ内の辺は、前記現在のスキャンラインの処理の完了時に順序付けられることを特徴とする請求項３に記載の方法。

【請求項５】　前記第２及び第３バッファは結合されて、前記選択的処理に用いられる第４バッファのそれぞれの部分を形成し、それにより、前記部分からの順序付けられた辺が漸進的に比較され、前記処理に対する次の変を決定することを特徴とする請求項２の方法。

【請求項６】　前記現在ｉのスキャンラインの処理の完了時において、前記第２及び第３バッファの内容と前記第４バッファの内容をスワップすることを特徴とする請求項５に記載の方法。

【請求項７】　前記第２、第３、第４バッファの各々は、そのスタート位置及びエンド位置のポインタを含み、前記現在のスキャンラインの終了時に、バッファをスワップするために該ポインタをスワップすることを特徴とする請求項６に記載の方法。

【請求項８】　前記第１バッファは、メモリに形成された辺プールを備え、前記方法は、前記メモリに形成された複数のアクティブ辺レコードから、前記現在のスキャンラインのためのアクティブ辺値を決定し、前記アクティブ辺値の各々に対し、対応する辺レコードを前記辺プールへ送るステップを更に備えることを特徴とする請求項１に記載の方法。

【請求項９】　前記第２バッファは、前記メモリに形成される現在辺出力リストを備え、前記方法は、前記アクティブ辺値の各々に対して、前記辺プール内のレコードと前記辺レコードの対応する辺レコードを調べ、該辺レコードの順序付けられたものを前記現在辺リストに送るステップを更に備えることを特徴とする請求項８に記載の方法。

【請求項１０】　前記第３バッファは、前記メモリ内に形成された現在スピル出力リストを備え、前記方法は、前記現在のスピルリストへ順次に前記辺プールからの前記現在辺リストに並べることのできない辺レコードを送るステップを更に備えることを特徴とする請求項９に記載の方法。

【請求項１１】　前記方法は、前記スキャンラインの処理の完了時において、前記後続のスキャンラインの処理のために前記出力リストの対応する一つへ順次に前記辺プールからの辺レコードをフラッシュするステップと、
ここで、このフラッシュに際しては前記出力リストは現在辺入力リスト及び現在辺入力スピルリストとしてそれぞれ割り当てられるものであり、該後続のスキャンラインに対する前記アクティブ辺値を決定するために前記入力リストから順次に辺レコードを前記アクティブ辺レコードの対応する一つに転送するステップとを備えることを特徴とする請求項１０に記載の方法。

【請求項１２】　グラフィックオブジェクト描画システムにおいて、ラスター画素イメージを形成するべくグラフィックオブジェクトを処理する方法であって、該処理は、前記グラフィックオブジェクトの辺と前記ラスター画素イメージの現在のスキャンラインとの交差の順序を決定する第１処理を備え、
前記システムが、
現在のスキャンラインと後続のスキャンラインの各々に関する複数の辺レコードと、該辺レコードの各々は少なくとも対応するスキャンライン上の対応する辺の画素位置の値を保持し、前記現在の辺レコード及び後続の辺レコードの各々は、少なくとも主部分とスピル部分に分割され、少なくとも前記現在の辺レコードの主部分はラスター画素順に並べられ、
少なくとも一つの現在のアクティブ辺レコードと、
スピルアクティブ辺レコードと、
削除された所定数の辺レコードを含むプールとを備え、
前記方法が、
（ａ）前記現在の辺レコードの前記主部分及びスピル部分の各々からの第１の辺レコードを対応するアクティブ辺レコードに送るステップと、
（ｂ）前記ラスタ画素順において最も低い値を有するアクティブ辺レコードを決定し、現在の辺の値及びレコードとしてその値とレコードを出力するために、前記アクティブ辺レコードの値を比較するステップと、
（ｃ）前記現在の辺レコードを、前記後続のスキャンラインのための対応する辺の値で更新するステップと、

（ｄ）更新された辺の値を前記プール内の辺のああ揃いと比較するステップと、ここで、更新された辺の値が前記プール内の辺の値より小さい場合に、
（ｄａ）前記更新された現在の辺レコードが後続の辺レコードのスピル部分へ送られ、さもなければ、
（ｄｂ）（ｄｂａ）最小の辺値を有する辺レコードが前記プールから、前記後続の辺レコードの主部分の次のレコードへ送られ、
（ｄｂｂ）前記更新された辺レコードは、前記サブステップ（ｄｂａ）で設になった前記プールのレコードへ送られ、
（ｄｃ）更なる辺レコードが、現在の辺レコードの対応する部分から更新された辺レコードによって空にされたアクティブ辺レコードへ送られ、
（ｅ）前記プール内の前記レコードの各々が占領されるまで、ステップ（ｂ）乃至（ｄ）を繰り返すステップと、これによって前記プールの最小の辺値レコードが前記後続の辺レコードの前記主部分へ送られ、
（ｆ）前記現在のレコードの全てのレコードが更新されるまで前記ステップ（ｂ）乃至（ｅ）を繰り返し、次いで、前記プールからのレコードを順次に前記後続のレコードの前記主部分内の次のレコードへフラッシュするステップと、
（ｇ）前記レコードを、前記後続のレコードにおける前記スピル部分において、ラスタ画素順にソートするステップと、
（ｈ）前記後続の辺レコードを前記現在の辺レコードへ送るステップと、
（ｉ）前記ラスタ画素イメージの更なるスキャンラインの各々に関して、前記ステップ（ａ）乃至（ｈ）を繰り返すステップとを備えることを特徴とする方法。
【請求項１３】　前記ステップ（ｇ）は、ソフトウエアソーティングルーチンにより実行されることを特徴とする請求項１２に記載の方法。
【請求項１４】　前記ステップ（ｄａ）が実行されたときに、前記ステップ（ｇ）が実行されることを特徴とする請求項１２に記載の方法。
【請求項１５】　前記ステップ（ｈ）は、前記現在の辺レコードを指すメモリロケーションと、後続の辺レコードを指すメモリロケーションをスワップすることにより実行されることを特徴とする請求項１２に記載の方法。
【請求項１６】　前記アクティブ辺レコードは、対応する新しい辺レコードから、現在のスキャンラインで開始する辺のレコードを受け取るための新たなアクティブ辺レコードを更に含み、前記ステップ（ｂ）は、前記新たな辺レコード、現在の辺レコード、スピル辺レコードの各々を比較して、最低の値を持つものをもって前記決定を行うことを特徴とする請求項１２に記載の方法。
【請求項１７】　前記ステップ（ｄ）において、前記変更された辺の値は、直後に変更され最後に前記プールへ加えられた辺値と比較されることを特徴とする請求項１２に記載の方法。
【請求項１８】　ラスター画素イメージを形成するべく、グラフィックオブジェクトを処理する装置であって、該処理は、ラスタライズされた表示順における現在のスキャンラインのための対応する辺レコードを評価することにより、前記グラフィックオブジェクトの辺間の交差順序を判断し、後続のスキャンラインのための各辺に関する辺交差値を決定するものであって、
未整列の第１バッファ、第２バッファ及び第３バッファを有するメモリと、
限られた数の処理された辺レコードを順序付けされていない前記第１バッファに保持し、順序付け可能な処理済の辺レコードが前記第１バッファに加えられのに応じて順序付け可能な前記辺レコードを漸進的に順次に前記第２バッファへ送り、順序付けできない処理済の辺を前記第３バッファにおいて並べるために該第３バッファへ送り、前記第２及び第３バッファからの辺レコードを選択的に処理し、後続のスキャンラインに対する並べられた交差を決定するプロセッサとを備えることを特徴とする装置。
【請求項１９】　前記第１バッファは、メモリに形成された辺プールを備え、前記プロセッサは、前記メモリに形成された複数のアクティブ辺レコードから、前記現在のスキャンラインのためのアクティブ辺値を決定し、前記アクティブ辺値の各々に対し、対応する辺レコードを前記辺プールへ送るように構成されることを特徴とする請求項１８に記載の装置。
【請求項２０】　前記第２バッファは、前記メモリに形成される現在辺出力リストを備え、前記プロセッサは、前記アクティブ辺値の各々に対して、前記辺プール内のレコードと前記辺レコードの対応する辺レコードを調べ、該辺レコードの順序付けられたものを前記現在辺リストに送るように構成されることを特徴とする請求項１９に記載の装置。
【請求項２１】　前記第３バッファは、前記メモリ内に形成された現在スピル出力リストを備え、前記プロセッサは、前記現在のスピルリストへ順次に前記辺プールからの前記現在辺リストに並べることのできない辺レコードを送るように構成されることを特徴とする請求項２０に記載の装置。
【請求項２２】　前記プロセッサは、前記スキャンラインの処理の完了時において、前記後続のスキャンラインの処理のために前記出力リストの対応する一つへ順次に前記辺プールからの辺レコードをフラッシュし、ここで、このフラッシュに際しては前記出力リストは現在辺入力リスト及び現在辺入力スピルリストとしてそれぞれ割り当てられるものであり、該後続のスキャンラインに対する前記アクティブ辺値を決定するために前記入力リストから順次に辺レコードを前記アクティブ辺レコード

の対応する一つに転送するように構成されることを特徴とする請求項２１に記載の方法。

【請求項２３】　ラスター画素イメージを形成するべく、グラフィックオブジェクトを処理する装置であって、該処理は、ラスタライズされた表示順における現在のスキャンラインのための対応する辺レコードを評価することにより、前記グラフィックオブジェクトの辺間の交差順序を判断し、後続のスキャンラインのための各辺に関する辺交差値を決定するものであって、

限られた数の処理された辺レコードを順序付けされていない第１バッファに保持し、順序付け可能な処理済の辺レコードが前記第１バッファに加えられのに応じて、順序付け可能な前記辺レコードを漸進的に順次に第２バッファへ送る手段と、

順序付けできない処理済の辺を、第３バッファにおいて並べるために該第３バッファへ送る手段と、

前記第２及び第３バッファからの辺レコードを選択的に処理し、後続のスキャンラインに対する並べられた交差を決定する手段とを備えることを特徴とする装置。

【請求項２４】　ラスター画素イメージを形成するべくグラフィックオブジェクトを処理するグラフィックオブジェクト描画システムの一部を形成する装置であって、

該処理は、前記グラフィックオブジェクトの辺と前記ラスター画素イメージの現在のスキャンラインとの交差の順序を決定する第１処理を備え、

前記装置が、

現在のスキャンラインと後続のスキャンラインの各々に関する複数の辺レコードと、該辺レコードの各々は少なくとも対応するスキャンライン上の対応する辺の画素位置の値を保持し、前記現在の辺レコード及び後続の辺レコードの各々は、少なくとも主部分とスピル部分に分割され、少なくとも前記現在の辺レコードの主部分はラスター画素順に並べられ、

少なくとも一つの現在のアクティブ辺レコードと、

スピルアクティブ辺レコードと、

制限された所定数の辺レコードを含むプールと、

辺レコード処理のために、

（ａ）前記現在の辺レコードの前記主部分及びスピル部分の各々からの第１の辺レコードを対応するアクティブ辺レコードに送り、

（ｂ）前記ラスタ画素順において最も低い値を有するアクティブ辺レコードを決定し、現在の辺の値及びレコードとしてその値とレコードを出力するために、前記アクティブ辺レコードの値を比較し、

（ｃ）前記現在の辺レコードを、前記後続のスキャンラインのための対応する辺の値で更新し、

（ｄ）更新された辺の値を前記プール内の辺のああ痛いと比較し、ここで、更新された辺の値が前記プール内の辺の値より小さい場合に、

（ｄａ）前記更新された現在の辺レコードが後続の辺レコードのスピル部分へ送られ、さもなければ、

（ｄｂ）（ｄｂａ）最小の辺値を有する辺レコードが前記プールから、前記後続の辺レコードの主部分の次のレコードへ送られ、

（ｄｂｂ）前記更新された辺レコードは、前記サブステップ（ｄｂａ）で競になった前記プールのレコードへ送られ、

（ｄｃ）更なる辺レコードが、現在の辺レコードの対応する部分から更新された辺レコードによって空にされたアクティブ辺レコードへ送られ、

（ｅ）前記プール内の前記レコードの各々が占領されるまで、ステップ（ｂ）乃至（ｄ）を繰り返し、これによって前記プールの最小の辺値レコードが前記後続の辺レコードの前記主部分へ送られ、

（ｆ）前記現在のレコードの全てのレコードが更新されるまで前記ステップ（ｂ）乃至（ｅ）を繰り返し、次いで、前記プールからのレコードを順次に前記後続のレコードの前記主部分内の次のレコードへフラッシュし、

（ｇ）前記レコードを、前記後続のレコードにおける前記スピル部分において、ラスタ画素順にソートし、

（ｈ）前記後続の辺レコードを前記現在の辺レコードへ送り、

（ｉ）前記ラスタ画素イメージの更なるスキャンラインの各々に関して、前記ステップ（ａ）乃至（ｈ）を繰り返す構成を有することを特徴とする装置。

【請求項２５】　ラスター画素イメージを形成するべく、グラフィックオブジェクトを処理する装置のためのプログラムを格納するコンピュータ可読媒体であって、該処理は、ラスタライズされた表示順における現在のスキャンラインのための対応する辺レコードを評価することにより、前記グラフィックオブジェクトの辺間の交差順序を判断し、後続のスキャンラインのための各辺に関する辺交差値を決定するものであって、該プログラムが、

限られた数の処理された辺レコードを順序付けされていない第１バッファに保持し、順序付け可能な処理済の辺レコードが前記第１バッファに加えられのに応じて、順序付け可能な前記辺レコードを漸進的に順次に第２バッファへ送る保持ステップのコードと、

順序付けできない処理済の辺を、第３バッファにおいて並べるために該第３バッファへ送る転送ステップのコードと、

前記第２及び第３バッファからの辺レコードを選択的に処理し、後続のスキャンラインに対する並べられた交差を決定する処理ステップのコードとを備えることを特徴とするコンピュータ可読媒体。

【請求項２６】　グラフィックオブジェクト描画システムにおいて、ラスター画素イメージを形成するべくグラフィックオブジェクトを処理するためのコンピュータプログラム製品を備えたコンピュータ可読媒体であって、

該処理は、前記グラフィックオブジェクトの辺と前記ラスター画素イメージの現在のスキャンラインとの交差の順序を決定する第１処理を備え、該コンピュータ製品は、
現在のスキャンラインと後続のスキャンラインの各々に関する複数の辺レコードと、該辺レコードの各々は少なくとも対応するスキャンライン上の対応する辺の画素位置の値を保持し、前記現在の辺レコード及び後続の辺レコードの各々は、少なくとも主部分とスピル部分に分割され、少なくとも前記現在の辺レコードの主部分はラスター画素順に並べられ、
少なくとも一つの現在のアクティブ辺レコードと、
スピルアクティブ辺レコードと、
制限された所定数の辺レコードを含むプールと関連し、前記第１処理が、
（ａ）前記現在の辺レコードの前記主部分及びスピル部分の各々からの第１の辺レコードを対応するアクティブ辺レコードに送るステップと、
（ｂ）前記ラスタ画素順において最も低い値を有するアクティブ辺レコードを決定し、現在の辺の値及びレコードとしてその値とレコードを出力するために、前記アクティブ辺レコードの値を比較するステップと、
（ｃ）前記現在の辺レコードを、前記後続のスキャンラインのための対応する辺の値で更新するステップと、
（ｄ）更新された辺の値を前記プール内の辺のああ痛いと比較するステップと、ここで、更新された辺の値が前記プール内の辺の値より小さい場合に、
（ｄａ）前記更新された現在の辺レコードが後続の辺レコードのスピル部分へ送られ、さもなければ、
（ｄｂ）（ｄｂａ）最小の辺値を有する辺レコードが前記プールから、前記後続の辺レコードの主部分の次のレコードへ送られ、
（ｄｂｂ）前記更新された辺レコードは、前記サブステップ（ｄｂａ）で空になった前記プールのレコードへ送られ、
（ｄｃ）更なる辺レコードが、現在の辺レコードの対応する部分から更新された辺レコードによって空にされたアクティブ辺レコードへ送られ、
（ｅ）前記プール内の前記レコードの各々が占領されるまで、ステップ（ｂ）乃至（ｄ）を繰り返すステップと、これによって前記プールの最小の辺値レコードが前記後続の辺レコードの前記主部分へ送られ、
（ｆ）前記現在のレコードの全てのレコードが更新されるまで前記ステップ（ｂ）乃至（ｅ）を繰り返し、次いで、前記プールからのレコードを順次に前記後続のレコードの前記主部分内の次のレコードへフラッシュするステップと、
（ｇ）前記レコードを、前記後続のレコードにおける前記スピル部分において、ラスタ画素順にソートするステップと、
（ｈ）前記後続の辺レコードを前記現在の辺レコードへ送るステップと、
（ｉ）前記ラスタ画素イメージの更なるスキャンラインの各々に関して、前記ステップ（ａ）乃至（ｈ）を繰り返すステップとを実行するように構成されてなることを特徴とするコンピュータ可読媒体。

【請求項２７】　前記処理は、前記イメージの部分を形成する少なくとも一つの前記グラフィックオブジェクトにより定義されるピクセルイメージデータを再生するための（第２の）処理を備え、該イメージはラスタ化方式で描画され、前記方法が、
（ｊ）各タイプの少なくとも一つの特性によって区別が可能な、複数のグラフィックオブジェクト塗り潰しタイプを確立するステップと、
（ｋ）描画を要求するグラフィックオブジェクトを受け取り、対応する塗り潰しタイプを決定し、該決定された塗り潰しタイプを用いて前記グラフィックオブジェクトに関連する塗り潰しレコードをアクセスするステップと、
（ｌ）前記塗り潰しレコードにか飛遍する塗り潰し処理を実行し、前記グラフィックオブジェクトに対応するラスタ化方式における前記画素イメージを決定することを特徴とする請求項１または１２に記載の方法。

【請求項２８】　前記タイプの一つがラスタイメージタイプであり、前記オブジェクトがラスタイメージタイプオブジェクトを特定する塗り潰しタイプを有する場合、前記ステップ（ｉ）は、前記画素イメージデータへのマッピングを決定するために前記イメージ中の画素位置を用いるステップを備え、これにおいて、そのようにマップされたデータは前記画素位置での出力のために選択されることを特徴とする請求項２７に記載の方法。

【請求項２９】　前記マッピングは、前記ピクセルイメージデータのアフィン変換を含むことを特徴とする請求項２７に記載の方法。

【請求項３０】　前記マッピングは、前記ラスタイメージに対して画素イメージデータのテクスチャマッピングを実行することを特徴とする請求項２７に記載の方法。

【請求項３１】　描画されるべき前記ラスタイメージにおける画素位置はテーブル内のエントリへのアクセスに用いられ、該テーブルは、前記グラフィックオブジェクト内で描画されるべき前記画素イメージデータの成分への参照を含み、前記第２処理は、前記画素イメージデータの部分をアクセスするために前記参照を操作し、該画素イメージデータは前記画素位置における前記ラスタイメージの描画された画素値へ寄与するのに用いられることを特徴とする請求項２７に記載の方法。

【請求項３２】　前記画素イメージデータの寄与は、前記ラスタイメージにおいて再生される単一の画素データ値を備えることを特徴とする請求項３１に記載の方法。

【請求項３３】　前記画素イメージデータの寄与は少な

くとも一つの画素データ値を備え、該画素データは、前記ラスタ化イメージにおいて再生される単一の画素値を派生するために、該ラスタイメージ内の少なくとも一つの他のグラフィックオブジェクトから派生した少なくとも一つの値と合成されることを特徴とする請求項３１に記載の方法。

【請求項３４】　前記画素イメージデータの寄与は一つの画素データ値を備え、該画素データは、前記ラスタ化イメージにおいて再生される複数の画素値を派生するために、該ラスタイメージ内の少なくとも一つの他のグラフィックオブジェクトから派生した少なくとも一つの値と合成されることを特徴とする請求項３１に記載の方法。

【請求項３５】　前記マッピングは、前記画素イメージデータ内のマッピングされた位置の周りに配置された複数の画素イメージデータを補間することを含むことを特徴とする請求項２８に記載の方法。

【請求項３６】　前記ステップ（ｋ）は、

（ｋａ）オブジェクトの特性に従って並べられた合成した特性のテーブルから得られるデータより、前記ラスタイメージのスキャンライン上の前記（異なる）画素の各々に対応する値のスタックを展開するサブステップを備え、ここで、該スタックは該画素値でアクティブなオブジェクトの数に対応する深さを有しており、

前記ステップ（ｉ）は、

（ｉａ）対応するスタックを用いて前記画素位置でのピクセル値を決定するサブステップを備えることを特徴とする請求項２７に記載の方法。

【請求項３７】　前記グラフィックオブジェクトの各々は、前記オブジェクトの境界の少なくとも一部を定義する少なくとも一つの辺を備えたデータ構造によって記述され、前記辺は、線分記述によって記述される複数の線分を含み、前記辺の開始及び終了するスキャンライン間で延び、少なくとも２つの部分が異なるデータフォーマットにより定義されることを特徴とする請求項１または１２に記載の方法。

【請求項３８】　前記線分どうしの接続において、前記線分の一つの最初の描画により返される値が、前記線分の一つに続くもののために初期決定値として用いられることを特徴とする請求項３７に記載の方法。

【請求項３９】　前記データフォーマットの各々は、前記線分のスタート及びエンドを示す識別子を含み、前記線分の確定を許容する更なるパラメータを含むことを特徴とする請求項３７に記載の方法。

【請求項４０】　前記線分は、直行するステップ線分であり、前記更なるパラメータは符号付のＸステップ値と、符号無しのＹステップ値を含むことを特徴とする請求項３９に記載の方法。

【請求項４１】　前記線分は、スキャンライン方向に対して傾斜を有する直線であり、前記更なるパラメータは、続くスキャンラインに対する画素位置の値を派生するために、現在のスキャンラインの画素位置の値に加える１次の値を含むことを特徴とする請求項３９に記載の方法。

【請求項４２】　前記線分は２次曲線であり、前記更なるパラメータは、次のスキャンラインに対する画素位置の値を派生するために、現在のスキャンラインの画素位置の値に加算するための前記現在のスキャンラインに関連する１次の値と、次のスキャンラインにおける１次の値を派生するために前記現在のスキャンラインの一次の値に加算される２次の値を含むことを特徴とする請求項３９に記載の方法。

【請求項４３】　前記線分はＮ次の多項式曲線であり、前記更なるパラメータは、現在のスキャンラインの画素位置の値を決める現在のスキャンラインに関連する１～Ｎ次の値を含み、これらは、次のスキャンラインの画素位置を派生するための次のスキャンラインの１～Ｎ－１次の値を派生するために加算されることを特徴とする請求項３９に記載の方法。

【請求項４４】　前記処理は線分の辺を描画する第３処理を備え、前記方法は複数の線分データタイプを解釈し、一つの線分から、後続の線分に対する画素位置を決定するのに用いられる第１の線分の画素位置を終了させることから生じる隣接する線分へのシームレスな遷移によって特徴付けられる対応するデータタイプで操作することにより、前記線分の各々に関して前記辺上の画素の位置を評価するステップを備えることを特徴とする請求項１または１２に記載の方法。

【請求項４５】　前記処理は、各々が対応する色と不透明度を有する第１と第２のピクセルを合成するための第４の処理を備え、該第４の処理が、

（ｊ）前記各画素値を少なくとも２つの領域に分け、第１領域を完全に不透明な領域とし、他を完全に透明な領域とするステップと、

（ｋ）全貴重いう行きの各々の間の交差する領域を決定し、交差領域の各々に対する不透明度値を派生するステップと、

（ｌ）合成された画素値に関して色と不透明度とを寄与する少なくとも一つの領域を選択するステップと、

（ｍ）所定のラスタ演算にしたがって、選択された領域の色を決定するステップと、

（ｎ）選択された領域に対する色－不透明度の積の和により第１の和を形成し、該第１の和を選択された領域の不透明度の和で形成される第２の和で除算して、該合成画素の色値を派生し、該第２の和を該合成画素の不透明度とするステップとを備えることを特徴とする請求項１または１２に記載の方法。

【請求項４６】　前記第１或いは第２画素値のうちの一方の領域は、前記第２或いは大１画素値の他方の領域と互いに直交し、ステップ（ｌ）～（ｎ）が適用される対

象となる３つの交差領域を定義することを特徴とする請求項４５に記載の方法。

【請求項４７】　前記合成演算は、前記対象となる領域の少なくとも一つを特定し、該特定された対象となる領域を前記和において用いることを特徴とする請求項４６に記載の方法。

【請求項４８】　前記ステップ（ｊ）～（ｌ）のいずれかが、前記色と不透明度値を正規化するステップを備えることを特徴とする請求項４５に記載の方法。

【請求項４９】　前記ステップ（ｍ）は、対応する前記領域に対する線引きするために、正規化された不透明度値を用いることを含むことを特徴とする請求項４８に記載の方法。

【請求項５０】　前記処理は、元側画素の色と不透明度値（ｓｏ）と、目的側画素の色と不透明度値（ｄｏ）を合成するための第４処理を備え、該第４処理は、

（ｊ）前記元側画素と目的側画素の各々の色と不透明度値を正規化して、完全に不透明な領域｛（ｓｏ），（ｄｏ）｝と、他の完全に透明な領域｛（１－ｓｏ），（１－ｄｏ）｝のうちの少なくとも２つの領域を定義するステップと、

（ｋ）目的側画素の領域が元画素の領域に直行して交差し、３つの交差領域の各々に対して、

（ｉ）目的側画素の外の元側画素｛ｓｏ×（１－ｄｏ）｝

（ii）元側と目的側画素の交差部｛ｓｏ×ｄｏ｝

（iii）元側画素の外の目的側画素｛（１－ｓｏ）×ｄｏ｝

なる不透明度値を派生するステップと、

（ｌ）元側画素と目的側の交差色（ｓｃ×ｄｃ）値を所定のラスタ演算に従って合成された画素値の不透明度成分として決定するステップと、

（ｍ）ｒｏｐが前記所定のラスタ演算を表すものとして、ｓｃ（ｓｏ×（１－ｄｏ））、（ｓｏ×ｄｏ）（ｓｃ　ｒｏｐ　ｄｃ）およびｄｃ（（１－ｓｃ）×ｄｏ）で表される選択された色－不透明度の積の和を用いて、前記合成された画素値の不透明度成分を決定することを特徴とする請求項１または１２に記載の方法。

【請求項５１】　前記色－不透明度の積は、所定の不透明度合成演算に従って選択可能であることを特徴とする請求項５０に記載の方法。

【請求項５２】　前記所定の不透明度合成演算は、前記交差領域の各々に対応するフラグを備え、前記和は、対応する前記フラグがセットされている前記交差領域のエリアの合計として決定されることを特徴とする請求項５１に記載の方法。

【請求項５３】　前記処理は、各々が対応する色と不透明度値を有する第１及び第２の画素値を所定の合成演算に従って合成し、これにより合成された画素値を形成する第４処理を備え、該第４処理が、

（ｊ）前記各画素値を少なくとも２つの領域に分け、第１領域を完全に不透明な領域とし、他を完全に透明な領域とするステップと、

（ｋ）全真虚いう行きの各々の間の交差する領域を決定し、交差領域の各々に対する不透明度値を派生するステップと、

（ｌ）合成された画素値に関して色と不透明度とを有する少なくとも一つの領域を前記合成演算に従って選択するステップと、

（ｍ）所定のラスタ演算にしたがって、選択された領域の各々の色を決定するステップと、

（ｎ）前記合成演算の一部である所定の不透明度演算に従って、選択された領域に対する色－不透明度の積の和により第１の和を形成し、該第１の和を選択された領域の不透明度の和で形成される第２の和で除算して、該合成画素値の色値を派生し、該第２の和を該合成画素の不透明度とするステップとを備え、前記不透明度演算は前記領域の各々に対応するフラグを備え、前記第１の和は前記フラグがセットされた領域のエリアの和として決定されることを特徴とする請求項１または１２に記載の方法。

【請求項５４】　前記オブジェクトの各々の透明度成分についてアカウントするオブジェクト間のラスター演算により特徴付けられるグラフィックオブジェクトの合成を更に備えることを特徴とする請求項１または１２に記載の方法。

【請求項５５】　前記処理は、前記グラフィカルオブジェクト環境に合成表現を適用する第５処理を備え、前記合成表現は複数の優先レベルを有し、該第５処理は、各優先レベルに関して、昇順の優先順で、

前記優先レベルの少なくとも一つのオペランドに関連する演算から発生するのに有用な複数の代替アクションを定義するステップと、

前記優先レベルに対する複数のアクティベーション条件に前記代替アクションを関連させるステップと、該アクティベーション条件は各オペランドのアクティビティ状態を含み、

次に高い優先レベルへの決定と応用のために前記代替アクションの一つを選択するべく前記アクティベーション条件を論理的に結合するステップとを備えることを特徴とする請求項１または１２に記載の方法。

【請求項５６】　前記アクティベーション条件は、対応する条件を示す少なくとも一つのフラグを含み、前記演算は、データと、該データを用いる後続の演算のためのエントリへのポインタと、どのオペランドが該データを提供するかを示す少なくとも一つのフラグとを前記対応する条件に関して提供することを特徴とする請求項５５に記載の方法。

【請求項５７】　前記合成表現は、昇順の優先順において選択された代替アクションを描画することにより評価

されることを特徴とする請求項５５に記載の方法。

【請求項５８】　前記代替アクションの各々はスタック演算を備え、該スタック演算は、互いに、スキャンライン上の２つのオブジェクトの辺の間の画素のスパンに関する画素出力値を決定することを特徴とする請求項５７に記載の方法。

【請求項５９】　前記合成表現は、前記イメージの階層的に構造化された表現であることを特徴とする請求項５５に記載の方法。

【請求項６０】　前記合成表現は、該合成表現によって表された前記イメージを描画するのに要求される画素演算の数に関して最適化されることを特徴とする請求項５８に記載の方法。

【請求項６１】　前記アクティベーション条件の論理的な結合は、前記階層的に構造化された表現を改造することなく、前記合成表現が評価される方式を改造することを備えることを特徴とする請求項６０に記載の方法。

【請求項６２】　前記イメージは、少なくとも画素ベースのイメージ成分を備えることを特徴とする請求項５９に記載の方法。

【請求項６３】　特定の優先レベルで、完全に不透明なグラフィックオブジェクトは、一つまたは複数の、前記合成表現におけるより低い優先レベルのオブジェクトを排除するべく動作することを特徴とする請求項５５に記載の方法。

【請求項６４】　前記ステップの少なくとも一つが、前記代替アクションと前記アクティベーション条件のテーブルを形成することを含むことを特徴とする請求項５４に記載の方法。

【請求項６５】　前記処理は、最適化された合成表現を形成する第５処理を備え、前記合成表現は前記グラフィックオブジェクトの階層的に構造化された表現であり、前記グラフィックオブジェクトの各々は所定のアウトラインを有し、該第５処理が、

複数の領域に対する表現を決定するステップと、前記領域の各々は前記所定のアウトライン或いはその部分の少なくとも一つに実質的に続く少なくとも一つの領域アウトラインにより定義され、

前記領域の少なくとも一つに対する少なくとも一つの特性に依存して、更なる複数の領域を特定するステップと、ここで、該更なる領域の各々は、関連付けられた合成演算を有し、

前記更なる領域と前記関連付けられた合成演算とを論理的に結合するステップと、

前記論理的結合を、前記最適化された合成表現を形成するために、前記階層的に構造化された表現のより高いレベルに適用するステップとを備えることを特徴とする請求項１または１２に記載の方法。

【請求項６６】　前記ステップの少なくとも一つが、前記代替アクションと前記アクティベーション条件のテーブルを形成することを含むことを特徴とする請求項６５に記載の方法。

【請求項６７】　前記処理は、前記イメージの描画のために用いられる合成演算のスタックを形成する第５処理を含み、該第５処理が、

前記グラフィカルオブジェクトの各々が再生可能な優先レベルに従って並べられたテーブルを確立するステップと、ここで、該テーブルは前記優先レベルに対して、

（ｉ）前記優先レベルで少なくとも一つのオペランドと関連する演算から生じるのに有用な複数の代替アクションのためのエントリと、

（ｉｉ）全規格オペランドに対するアクティビティ状態を含むアクティビティ条件のためのエントリとを含み、

表示可能なスキャンライン上の２つの隣り合うオブジェクトの辺の間の表示可能な画素のスパンに対する前記エントリと各スパンのためのエントリとを更新するステップと、

前記スタックへ昇順の優先度順で送るための前記優先レベルの各々に関し、前記代替アクションの一つを選択するために、前記アクティベーション条件を解析するステップとを備えることを特徴とする請求項１または１２に記載の方法。

【請求項６８】　前記処理は、複数のグラフィックオブジェクトで形成された前記イメージを描画するのに使用する合成スタックを生成する第６処理を備え、該グラフィックオブジェクトの各々は、少なくとも辺と、他のグラフィックオブジェクトに対するビューイングの優先度とを含む成分により記述され、該第６処理が、

（ｊ）前記オブジェクトの境界とそれらの交差を決定するステップと、

（ｋ）決定された前記交差の各々について、ビューイングの優先度に従って並べられた前記交差においてアクティブなオブジェクトのレベルアクティベーションテーブルを提供するステップと、

（ｌ）アクティブな上側に配されるグラフィックオブジェクトが、重複領域内で下側に配されるグラフィックオブジェクトに重複するかどうかを確かめるステップと、

（ｍ）そのような重複が生じた場合に、上側に配されるグラフィックオブジェクトと下側に配されるグラフィックオブジェクトの境界を用いてクリッピング演算を実行し、重複領域内の上側に配置されるグラフィックオブジェクト及び／又は下側に配されるグラフィックオブジェクトのクリップバージョン、及び／又は重複領域外の下側に配されるオブジェクト及び／又は上側に配されるグラフィックオブジェクトのクリップバージョンを生成するステップと、

（ｎ）レベルアクティベーションテーブルに、

（ｎａ）重複領域内へ配置された、下側及び上側に配されるグラフィックオブジェクトの領域に対応する一つ又は複数の第１のレベル、及び／又は

（ｎｂ）重複領域外へ配置された、下側及び／又は上側に配されるグラフィックオブジェクトの領域に対応する一つ又は複数の第２のレベルを加えるステップと、

（ｏ）前記イメージ内の各ピクセルに対し、当該ピクセルに適切なレベルアクティベーションテーブルのエントリに基づいて、合成スタックを決定するステップとを備えることを特徴とする請求項１または１２に記載の方法。

【請求項６９】　前記ステップ（ｊ）は、所定の順序で、現在のスキャンラインと交差する前記オブジェクトの辺の交差座標を決定するサブステップ（ｊａ）を備えることを特徴とする請求項６８に記載の方法。

【請求項７０】　前記ステップ（ｌ）が、重複するグラフィックオブジェクトへ適用する合成演算子を決定するサブステップ（ｌａ）を備え、

前記ステップ（ｍ）が、前記サブステップ（ｌａ）で決定された合成演算子に基づいて、上側に配されるオブジェクト及び／又は下側に配されるオブジェクトの辺を用いて、クリッピング演算を実行するサブステップ（ｍａ）を備えることを特徴とする請求項６８に記載の方法。

【請求項７１】　前記サブステップ（ｎａ）が、下側に配される及び／又は上側に配されるオブジェクトに対する一つ又は複数の前記第２レベルを前記レベルアクティベーションテーブルに加え、該一つ又は複数の第２レベルは、前記サブステップ（ｌａ）で決定された合成演算子に基づいた該レベルアクティベーションテーブルへの追加のために選択されることを特徴とする請求項７０に記載の方法。

【請求項７２】　前記合成演算子は、ポータアンドダフ合成演算子であることを特徴とする請求項６９に記載の方法。

【請求項７３】　前記処理は、複数のグラフィックオブジェクトで形成された前記イメージを描画するのに使用する合成スタックを、レベルアクティベーションテーブルにおけるエントリに基づいて生成する第６処理を備え、該グラフィックオブジェクトの各々は、少なくとも辺と、他のグラフィックオブジェクトに対するビューイングの優先度とを含む成分により記述され、該第６処理が、

（ｊ）所定の順序で、前記スキャンラインと交差する前記オブジェクトの辺の交差座標を決定するステップと、

（ｋ）決定された前記辺の交差の各々について、前記レベルアクティベーションテーブルを更新するステップと、該レベルアクティベーションテーブルはビューイングの優先度に従って並べられており、

（ｌ）アクティブな上側に配されるグラフィックオブジェクトが、重複領域内で下側に配されるグラフィックオブジェクトに重複するかどうかを確かめるステップと、

（ｍ）そのような重複が生じた場合に、上側に配されるグラフィックオブジェクトと下側に配されるグラフィックオブジェクトの辺を用いてクリッピング演算を実行し、重複領域内の上側に配置されるグラフィックオブジェクト及び／又は下側に配されるグラフィックオブジェクトのクリップバージョン、及び／又は重複領域外の下側に配されるオブジェクト及び／又は上側に配されるグラフィックオブジェクトのクリップバージョンを生成するステップと、

（ｎ）レベルアクティベーションテーブルに、

（ｎａ）重複領域内へ配置された、下側及び上側に配されるグラフィックオブジェクトの領域に対応する一つ又は複数の第１のレベル、及び／又は

（ｎｂ）重複領域外へ配置された、下側及び／又は上側に配されるグラフィックオブジェクトの領域に対応する一つ又は複数の第２のレベルを加えるステップと、

（ｏ）スキャンライン上の各ピクセル位置に対し、当該画素位置に適切なレベルアクティベーションテーブルのエントリに基づいて、合成スタックを決定するステップとを備えることを特徴とする請求項１または１２に記載の方法。

【請求項７４】　前記ステップ（ｍ）が、上側に配されるグラフィックオブジェクトと下側に配されるグラフィックオブジェクトの辺を用いてクリッピング演算を実行し、重複領域内の上側及び下側に配されるグラフィックオブジェクトのクリップバージョンと、該重複領域外の下側或いは上側に配されるグラフィックオブジェクトのクリップバージョンとを生成することを特徴とする請求項７３に記載の方法。

【請求項７５】　前記重複するオブジェクトは、非不透明であることを特徴とする請求項６８に記載の方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、オブジェクト・グラフィック要素のラスタ画素画像へのレンダリングに関し、具体的には、レンダリング処理の一部としての画素データのフレーム記憶装置またはライン記憶装置を使用しない、そのようなオブジェクト・グラフィック要素の画素画像データへの効率的なレンダリングに関する。

【０００２】

【従来の技術】ほとんどのオブジェクト・ベース・グラフィックス・システムでは、ページまたは画面の画素ベース画像を保持するためにフレーム・ストアまたはページ・バッファが使用される。通常、グラフィック・オブジェクトの輪郭は、計算され、塗り潰され、フレーム・ストアに書き込まれる。二次元グラフィックスの場合、他のオブジェクトの手前にあるオブジェクトは、単純に背景オブジェクトの書込み後にフレーム・ストアに書き込まれ、これによって、画素単位で背景を置換する。これは、当技術分野では、「ペインタのアルゴリズム（Painter's algorithm）」として一般に知られている。オ

ブジェクトは、最も奥のオブジェクトから最も手前のオブジェクトという優先順位で検討され、通常は、各オブジェクトが、スキャン・ラインの順序でラスタ化され、画素が、各スキャン・ラインに沿ったシーケンシャルな並びでフレーム・ストアに書き込まれる。

【０００３】この技法には、基本的に２つの問題がある。第１の問題は、フレーム・ストア内のすべての画素への高速なランダム・アクセスが必要であることである。これは、新たに検討されるオブジェクトのそれぞれが、フレーム・ストア内のどの画素にも影響する可能性があるからである。このため、フレーム・ストアは、通常は半導体のランダム・アクセス・メモリ（ＲＡＭ）内に保持される。高解像度カラー・プリンタの場合、必要なＲＡＭの量は非常に多くなり、通常は１００Ｍバイトを超えるが、これは、コストがかかり、高速での動作が困難である。第２の問題は、多数の画素がペイント（レンダリング）され、時間的に後のオブジェクトによって上塗り（再レンダリング）されることである。時間的に前のオブジェクトによる画素のペイントは、時間の浪費になる。

【０００４】大量のフレーム・ストアの問題を克服するための方法の１つが、「バンディング（banding）」の使用である。バンディングを使用する時には、一時にフレーム・ストアの一部だけがメモリ内に存在する。描画されるオブジェクトのすべてが、「表示リスト」に保存される。画像全体は上記と同様にレンダリングされるが、存在するフレーム・ストアの部分の外側をペイント（レンダリング）しようとする画素ペイント（レンダリング）動作は、「クリップ」アウトされる。オブジェクトのすべてが描画された後に、フレーム・ストアの部分を、プリンタ（または他の位置）に送り、フレーム・ストアの別の部分を選択し、この処理を繰り返す。この技法には、ペナルティが伴う。たとえば、描画されるオブジェクトは検討され、何度も（バンドごとに１回）再検討されなければならない。バンドの数が増えるにつれて、レンダリングが必要なオブジェクトの検査が繰り返されるし回数も増える。バンディングの技法は、上塗りのコストという問題を解決しない。

【０００５】いくつかの他のグラフィック・システムでは、スキャン・ラインの順で画像が検討される。やはり、描画されるオブジェクトのすべてが、表示リストに保存される。各スキャン・ライン上で、そのスキャン・ラインと交差するオブジェクトが優先順位の順でオブジェクトごとに検討され、オブジェクトの辺の交差点の間の画素のスパンが、ライン・ストアにセットされる。この技法も、大量のフレーム・ストアの問題を克服するが、やはり上塗りの問題をこうむる。

【０００６】これらのほかに、大きいフレーム・ストアの問題と上塗りの問題の両方を解決する技法がある。そのような技法の１つでは、各スキャン・ラインが順番に作られる。やはり、描画されるオブジェクトのすべてが、表示リストに保存される。各スキャン・ラインでは、そのスキャン・ラインと交差するオブジェクトの辺が、スキャン・ラインとの交差の座標の昇順で保持される。これらの交差の点または辺の交点が、順番に検討され、アクティブ・フラグの配列のトグルに使用される。そのスキャン・ライン上での対象となるオブジェクト優先順位ごとに１つのアクティブ・フラグがある。検討される辺の対のそれぞれの間で、最初の辺と次の辺の間にある各画素の色データが、アクティブ・フラグに対する優先順位エンコーダを使用して、どの優先順位が最も上にあるかを判定すること、および２つの辺の間のスパンの画素に関するその優先順位に関連する色を使用することによって生成される。次のスキャン・ラインに備えて、各辺の交差の座標が、各辺の性質に応じて更新される。この更新の結果として誤ってソートされた状態になった隣接する辺は、交換される。新しい辺も、辺のリストに合併される。

【０００７】この技法は、フレーム・ストアまたはライン・ストアがなく、上塗りがなく、位数Ｎ倍（このＮは優先順位の数）ではなく定数位数の時間でオブジェクト優先順位が処理されるという大きい長所を有する。

【０００８】【発明が解決しようとする課題】しかし、この技法には下記の複数の制限がある。

【０００９】（ｉ）この技法は、辺からオブジェクトの内外の状態を決定するための、当技術分野で既知の「奇数－偶数」塗潰し規則だけをサポートする。多数のグラフィック記述言語の必須機能である「非ゼロ・ワインディング」塗潰し規則は、この技法によってサポートされない。

【００１０】（ｉｉ）単純な交換技法が修復に不適切である場合に、大量の躯ったソートが発生する可能性がある。各スキャン・ライン上で辺リスト全体のブルート・フォース・ソート（brute-force sort）を実行することができるが、これは非常に低速である。

【００１１】（ｉｉｉ）この技法は、オブジェクト・タイプとしてのラスタ（画素ベース）画像をサポートしない。このような画像は、ほとんどのグラフィック記述言語の必須機能である。

【００１２】（ｉｖ）この技法は、ペイントされる画素のそれぞれが、より低い優先順位のオブジェクトの画素を厳密に隠す、不透明のオブジェクトだけをサポートする。よって、この技法は、複数のグラフィック・オブジェクトの色が相互作用するラスタ演算をサポートしない。このような演算には、ＸＯＲモードでの描画または部分的に透明なオブジェクトの合成が含まれる。これらの変更演算は、ほとんどのグラフィック記述言語の必須機能である。

【００１３】（ｖ）この技法は、１つまたは複数のクリ

ップ形状によって、クリップ形状の境界の内側（または外側）にあるいくつかの他のグラフィック・オブジェクトを削除する、クリッピングをサポートしない。クリッピングは、ほとんどのグラフィック記述言語の必須機能である。

【００１４】（ｖｉ）この技法では、オブジェクトの辺の、具体的にはテキストに関する、大量の非効率的な符号化が使用される。このようなグラフィック記述の極端に一般的な要素は、より単純な形で表現されることが望ましい。

【００１５】（ｖｉｉ）この技法は、いくつかの場合に、１つまたは複数のオブジェクトのアクティビティが変数である複雑な合成式の正確な評価を提供しない。

【００１６】この技法は、既存のグラフィック記述言語が必要とする多数の機能を実施できないので、その使用が極度に制限されている。

【００１７】さらに、既存のレンダリング・インターフェースの中には、複数のグラフィック・オブジェクトの色のビット単位の論理組合せの実装を要求するものがあり、複数のグラフィック・オブジェクトの色のアルファ・チャネル（透明度、不透明度またはマットと称する）に基づく組合せの実装を要求するものもある。現在の技法では、この２つの機能を統一された形で実装することができない。

【００１８】

【課題を解決するための手段】本発明の目的は、従来技術のシステムに伴う１つ又は複数の欠陥を、実質的に克服するか、少なくとも改善することである。

【００１９】本発明の一つの態様によれば、ラスター画素イメージを形成するべく、グラフィックオブジェクトを処理する方法であって、該処理は、ラスタライズされた表示順における現在のスキャンラインのための対応する辺レコードを評価することにより、前記グラフィックオブジェクトの辺間の交差順序を判断し、後続のスキャンラインのための各辺に関する辺交差値を決定するものであって、該処理は前記辺レコードの処理の間に、限られた数の処理された辺レコードを順序付けされていない第１バッファに保持し、順序付け可能な処理済の辺レコードが前記第１バッファに加えられのに応じて、順序付け可能な前記辺レコードを漸進的に順次に第２バッファへ送るステップと、順序付けできない処理済の辺を、第３バッファにおいて並べるために該第３バッファへ送るステップと、前記第２及び第３バッファからの辺レコードを選択的に処理し、後続のスキャンラインに対する並べられた交差を決定するステップとを備える方法が開示される。

【００２０】また、本発明の他の態様によれば、グラフィックオブジェクト描画システムにおいて、ラスター画素イメージを形成するべくグラフィックオブジェクトを処理する方法であって、該処理は、前記グラフィックオブジェクトの辺と前記ラスター画素イメージの現在のスキャンラインとの交差の順序を決定する第１処理を備え、前記システムが、現在のスキャンラインと後続のスキャンラインの各々に関する複数の辺レコードと、該辺レコードの各々は少なくとも対応するスキャンライン上の対応する辺の画素位置の値を保持し、前記現在の辺レコード及び後続の辺レコードの各々は、少なくとも主部分とスピル部分に分割され、少なくとも前記現在の辺レコードの主部分はラスター画素順に並べられ、少なくとも一つの現在のアクティブ辺レコードと、スピルアクティブ辺レコードと、削限された所定数の辺レコードを含むプールとを備え、前記方法が、（ａ）前記現在の辺レコードの前記主部分及びスピル部分の各々からの第１の辺レコードを対応するアクティブ辺レコードに送るステップと、（ｂ）前記ラスタ画素順において最も低い値を有するアクティブ辺レコードを決定し、現在の辺の値及びレコードとしてその値とレコードを出力するために、前記アクティブ辺レコードの値を比較するステップと、（ｃ）前記現在の辺レコードを、前記後続のスキャンラインのための対応する辺の値で更新するステップ、（ｄ）更新された辺の値を前記プール内の辺のああ痛いと比較するステップと、ここで、更新された辺の値が前記プール内の辺の値より小さい場合に、（ｄａ）前記更新された現在の辺レコードが後続の辺レコードのスピル部分へ送られ、さもなければ、（ｄｂ）（ｄｂａ）最小の辺値を有する辺レコードが前記プールから、前記後続の辺レコードの主部分の次のレコードへ送られ、（ｄｂ）前記更新された辺レコードは、前記サブステップ（ｄｂａ）で殻になった前記プールのレコードへ送られ、（ｄｃ）更なる辺レコードが、現在の辺レコードの対応する部分から更新された辺レコードによって空にされたアクティブ辺レコードへ送られ、（ｅ）前記プール内の前記レコードの各々が占領されるまで、ステップ（ｂ）乃至（ｄ）を繰り返すステップと、これによって前記プールの最小の辺値レコードが前記後続の辺レコードの前記主部分へ送られ、（ｆ）前記現在のレコードの全てのレコードが更新されるまで前記ステップ（ｂ）乃至（ｅ）を繰り返し、次いで、前記プールからのレコードを順次に前記後続のレコードの前記主部分内の次のレコードへフラッシュするステップと、（ｇ）前記レコードを、前記後続のレコードにおける前記スピル部分においてラスタ画素順にソートするステップと、（ｈ）前記後続の辺レコードを前記現在の辺レコードへ送るステップと、（ｉ）前記ラスタ画素イメージの更なるスキャンラインの各々に関して、前記ステップ（ａ）乃至（ｈ）を繰り返すステップとを備える方法が開示される。

【００２１】また、本発明の他の態様によれば、ラスター画素イメージを形成するべく、グラフィックオブジェクトを処理する装置であって、該処理は、ラスタライズ

された表示順における現在のスキャンラインのための対応する辺レコードを評価することにより、前記グラフィックオブジェクトの辺間の交差順序を判断し、後続のスキャンラインのための各辺に関する辺交差値を決定するものであって、未整列の第１バッファ、第２バッファ及び第３バッファを有するメモリと、限られた数の処理された辺レコードを順序付けされていない前記第１バッファに保持し、順序付け可能な処理済の辺レコードが前記第１バッファに加えられのに応じて順序付け可能な前記辺レコードを漸進的に順次に前記第２バッファへ送り、順序付けできない処理済の辺を前記第３バッファにおいて並べるために該第３バッファへ送り、前記第２及び第３バッファからの辺レコードを選択的に処理し、後続のスキャンラインに対する並べられた交差を決定するプロセッサとを備える装置が開示される。

【００２２】また、本発明の他の態様によれば、ラスター画素イメージを形成するべく、グラフィックオブジェクトを処理する装置であって、該処理は、ラスタライズされた表示順における現在のスキャンラインのための対応する辺レコードを評価することにより、前記グラフィックオブジェクトの辺間の交差順序を判断し、後続のスキャンラインのための各辺に関する辺交差値を決定するものであって、限られた数の処理された辺レコードを順序付けされていない第１バッファに保持し、順序付け可能な処理済の辺レコードが前記第１バッファに加えられのに応じて、順序付け可能な前記辺レコードを漸進的に順次に第２バッファへ送る手段と、順序付けできない処理済の辺を、第３バッファにおいて並べるために該第３バッファへ送る手段と、前記第２及び第３バッファからの辺レコードを選択的に処理し、後続のスキャンラインに対する並べられた交差を決定する手段とを備える装置が開示される。

【００２３】また、本発明の他の態様によれば、ラスター画素イメージを形成するべく、グラフィックオブジェクトを処理する装置のためのプログラムを格納するコンピュータ可読媒体であって、該処理は、ラスタライズされた表示順における現在のスキャンラインのための対応する辺レコードを評価することにより、前記グラフィックオブジェクトの辺間の交差順序を判断し、後続のスキャンラインのための各辺に関する辺交差値を決定するものであって、該プログラムが、限られた数の処理された辺レコードを順序付けされていない第１バッファに保持し、順序付け可能な処理済の辺レコードが前記第１バッファに加えられのに応じて、順序付け可能な前記辺レコードを漸進的に順次に第２バッファへ送る保持ステップのコードと、順序付けできない処理済の辺を、第３バッファにおいて並べるために該第３バッファへ送る転送ステップのコードと、前記第２及び第３バッファからの辺レコードを選択的に処理し、後続のスキャンラインに対する並べられた交差を決定する処理ステップのコードとを備えるコンピュータ可読媒体が開示される。

【００２４】本発明の更に他の態様は以下の説明から明らかとなる。

【００２５】

【発明の実施の形態】以下、添付の図面を参照して本発明の実施形態を説明する。なお、以下の実施形態の記載において、数学記号を下記のように表記する。

【表１】

以下では、

$[A]$	を「A」で表わし、
$\overline{A}$	をA‾で表わし、
$\overline{Dest\_Active}$	を(Dest_Active)‾と表わす

図１は、コンピュータ・グラフィック・オブジェクト画像のレンダリングおよびプレゼンテーションのために構成されたコンピュータ・システム１を概略的に示す図である。このシステムには、システム・ランダム・アクセス・メモリ（ＲＡＭ）３に関連するホスト・プロセッサ２が含まれ、システムＲＡＭ３には、不揮発性のハード・ディスク・ドライブ５または類似の装置と、揮発性の半導体ＲＡＭ４を含めることができる。システム１には、システム読取専用メモリ（ＲＯＭ）６も含まれ、システムＲＯＭ６は、通常は半導体ＲＯＭ７を基礎とし、多くの場合に、コンパクト・ディスク装置（ＣＤ－ＲＯＭ）８によって補足することができる。システム１には、ラスタ式に動作するビデオ表示装置（ＶＤＵ）またはプリンタなどの、画像を表示するための手段１０も組み込むことができる。

【００２６】システム１に関して上で説明した構成要素は、バス・システム９を介して相互接続され、ＩＢＭ ＰＣ／ＡＴタイプのパーソナル・コンピュータおよびそれから発展した構成、Ｓｕｎ Ｓｐａｒｃｓｔａｔｉｏｎｓおよび類似物など、当技術分野で周知のコンピュータ・システムの通常動作モードで動作可能である。

【００２７】やはり図１に図示されているように、画素シーケンシャル・レンダリング装置２０は、バス９に接続され、好ましい実施形態では、システム１からバス９を介して命令およびデータを供給されるグラフィック・オブジェクト・ベースの記述から導出される画素ベースの画像のシーケンシャル・レンダリングのために構成される。装置２０は、オブジェクト記述のレンダリングのためにシステムＲＡＭ３を使用することができるが、レンダリング装置２０は、通常は半導体ＲＡＭから形成され、専用のレンダリング・ストア配置３０と関連付けられることが好ましい。

【００２８】次に図２を参照すると、好ましい実施形態の機能データ流れ図が示されている。図２の機能流れ図は、オブジェクト・グラフィック記述１１から始まる。

このオブジェクト・グラフィック記述１１は、ホスト・プロセッサ２によって生成されるか、且つ/又は、システムＲＡＭ３内に記憶されるかシステムＲＯＭ６から導出され、グラフィック・オブジェクトのパラメータを記述するのに使用され、そこから画素ベース画像をレンダリングするために、画素シーケンシャル・レンダリング装置２０によって解釈され得る。たとえば、オブジェクト・グラフィック記述１１には、ディスプレイ上の１点から別の点まで横断する直線の辺（単純ベクトル）または、直交する線を含む複数の辺によって二次元オブジェクトが定義される直交辺フォーマットを含むいくつかのフォーマットで辺を有するオブジェクトを組み込むことができる。これ以外に、連続曲線によってオブジェクトが定義されるフォーマットも、適当であり、これらには、乗算を実行する必要なしに二次曲線を単一の出力空間内でレンダリングできるようにするいくつかのパラメータによって単一の曲線を記述できる二次多項式の線分を含めることができる。三次スプラインや類似物などのそれ以外のデータ・フォーマットを使用することもできる。オブジェクトには、多数の異なる辺タイプの混合物を含めることができる。通常、すべてのフォーマットに共通するのは、それぞれの線（直線であれ曲線であれ）の始点と終点の識別子であり、通常は、これらは、スキャン・ライン番号によって識別され、したがって、その曲線をレンダリングすることのできる特定の出力空間が定義される。

【００２９】たとえば、図１６Ａに、線分を適当に記述し、レンダリングするために、２つの線分６０１および６０２に分割する必要がある辺６００の従来技術の辺記述を示す。分割の必要が生じるのは、従来技術の辺記述が、二次式を介して簡単に計算されるが、変曲点６０４に適応することができないからである。したがって、辺６００は、それぞれ終点６０３および６０４または終点６０４および６０５を有する２つの別々の辺として扱われた。図１６Ｂに、終点６１１および６１２と制御点６１３および６１４によって記述される三次スプライン６１０を示す。このフォーマットでは、レンダリングのために三次多項式の計算が必要であり、したがって、計算時間がかかる。

【００３０】図１６Ｃに、好ましい実施形態に適用可能な辺の例を示す。好ましい実施形態では、辺は、単一の実体とみなされ、必要であれば、異なるフォーマットで記述できる辺の部分を示すために区分されるが、その具体的な目的は、各部分の記述の複雑さが最小限になるようにすることである。

【００３１】図１６Ｃの左側には、スキャン・ラインＡ～Ｍの間にまたがる単一の辺６２０が示されている。辺は、ｓｔａｒｔ＿ｘ、ｓｔａｒｔ＿ｙ、辺の次の線分を指すアドレスを含む１つまたは複数の線分記述、および、辺の終了に使用される最終線分を含む複数のパラメータによって記述される。好ましい実施形態によれば、辺６２０は、３つのステップ線分、１つのベクトル線分および１つの二次線分として記述することができる。ステップ線分は、単純に、ｘステップ値とｙステップ値を有するものとして定義される。図示の３つのステップ線分の場合、線分記述は［０，２］、［＋２，２］および［＋２，０］である。ｘステップ値は符号付きであり、これによってステップの向きが示されるが、ｙステップ値は、必ず、スキャン・ラインの値が増えるラスタ・スキャン方向であるから符号なしであることに留意されたい。次の線分は、通常はパラメータｓｔａｒｔ＿ｘ、ｓｔａｒｔ＿ｙ、ｆｉｎｉｓｈ＿ｙおよび傾斜（ＤＸ）を必要とするベクトル線分である。この例では、ベクトル線分が辺６２０の中間線分であるから、ｓｔａｒｔ＿ｘおよびｓｔａｒｔ＿ｙは、前の線分から生じるので、省略することができる。傾斜値（ＤＸ）は、符号付きであり、前のスキャン・ラインのｘ値に加算されて、現行スキャン・ラインのｘ値を与え、図示の例では、ＤＸ＝＋１である。次の線分は、二次線分であり、これは、ベクトル線分に対応する構造を有するが、さらに、やはり符号付きであり、線分の傾斜を変更するためにＤＸに加算される２階値（ＤＤＸ）も有する。

【００３２】図１６Ｃの右側の線分は、好ましい実施形態による三次曲線の例を示す図であり、これには上記の二次線分に対応する記述が含まれるが、線分の傾斜の変化の割合を変更するためにＤＤＸに加算される符号付きの３階値（ＤＤＤＸ）が追加されている。同様に、多数の他の階も実施することができる。

【００３３】上記から、辺の線分を記述する複数のデータ・フォーマットを処理する能力があると、複雑で計算コストの高い数学演算に頼らずに、辺の記述と評価を単純化することができることが明白である。これに対して、図１６Ａの従来技術のシステムでは、直交、ベクトルまたは二次のいずれであれ、すべての辺を二次形式で記述する必要があった。

【００３４】好ましい実施形態の動作を、図８に示された画像７８のレンダリングという単純な例に関して説明する。画像７８は、２つのグラフィカル・オブジェクト、具体的に言うと、不透明の赤色の長方形９０とその上にレンダリングされ、これによって長方形９０を部分的に隠す、部分的に透明の青色の三角形８０を含む。図からわかるように、長方形９０には、種々の画素位置（Ｘ）とスキャン・ライン位置（Ｙ）の間で定義された横の辺９２、９４、９６および９８が含まれる。辺９６および９８は、スキャン・ライン上に形成される（したがってこれらと平行である）ので、長方形９０の実際のオブジェクト記述は、図９Ａに示されているように、横の辺９２および９４だけに基づくものとすることができる。これに関連して、辺９２は、画素位置（４０，３５）から始まり、ラスタ方向で画面の下側へ延びて、画

素位置（４０，１０５）で終わる。同様に、辺９４は、画素位置（１６０，３５）から位置（１６０，１０５）まで延びる。長方形グラフィック・オブジェクト９０の水平部分は、単に辺９２から辺９４へラスタ化された形で走査することによって得ることができる。

【００３５】しかし、青い三角形のオブジェクト８０は、３つのオブジェクト辺８２、８４および８６によって定義され、各辺は、三角形の頂点を定義するベクトルとみなされる。辺８２および８４は、画素位置（１００，２０）から始まり、それぞれ画素位置（１７０，９０）または（３０，９０）まで延びる。辺８６は、これら２つの画素位置の間で、従来のラスタ化された左から右への方向に延びる。この特定の例では、辺８６が、上で述べた辺９６および９８と同様に水平なので、辺８６が定義されることは必要ではない。というのは、辺８６が、辺８２および８４に関する終点をもつという特徴があるからである。辺８２および８４の記述に使用される始点および終点の画素位置のほかに、これらの辺のそれぞれに、この場合ではそれぞれ＋１または－１の傾斜値が関連付けられる。

【００３６】図１０に、スキャン・ライン３５で始まる長方形９０がレンダリングされる様子と、辺８２および８４がスキャン・ライン３５とどのように交差するかを示す。図１０から、画像７８のラスタ化には、高い優先順位レベルを有するオブジェクトが、低い優先順位レベルを有するオブジェクトの「上」にレンダリングされる形で２つのオブジェクト９０および８０が描画される必要のあることがわかる。これを図１１に示す。図１１は、画像７８のレンダリングに使用される辺リスト・レコードを示す図である。図１１のレコードには、オブジェクトごとに１つずつの、２つの項目が含まれる。これらの項目は、それぞれのオブジェクトのラスタ・レンダリング順での始点に対応するスキャン・ライン値で配置される。図１１から、辺レコードのそれぞれが、オブジェクトの関連する優先順位レベルと、記述される辺の性質に関する詳細（たとえば色、傾斜など）を有することがわかる。

【００３７】再び図２に戻って説明する。レンダリングされるグラフィック・オブジェクトの記述に必要なデータを識別したので、グラフィック・システム１は、表示リスト生成ステップ１２を実行する。

【００３８】表示リスト生成１２は、取り付けられたＲＯＭ６およびＲＡＭ３を有するホスト・プロセッサ２上で実行されるソフトウェア・モジュールとして実施されることが好ましい。表示リスト生成１２では、周知のグラフィック記述言語、グラフィック・ライブラリ呼出しまたは他のアプリケーション固有フォーマットのうちの１つまたは複数で表現されたオブジェクト・グラフィック記述を表示リストに変換する。表示リストは、通常は、表示リスト・ストア１３に書き込まれる。表示リスト・ストア１３は、一般にＲＡＭ４内で形成されるが、その代わりにレンダリング・ストア３０内で形成することもできる。図３からわかるように、表示リスト・ストア１３には、複数の構成要素を含めることができ、その１つは命令ストリーム１４であり、もう１つは辺情報１５であり、ラスタ画像画素データ１６を含めることができる。

【００３９】命令ストリーム１４には、特定の画像内で所望される特定のグラフィック・オブジェクトをレンダリングするために画素シーケンシャル・レンダリング装置２０によって読み取られる、命令として解釈可能なコードが含まれる。図８に示された画像の例では、命令ストリーム１４が、下記の形になり得る。

（１）スキャン・ライン２０までレンダリングする（何もしない）

（２）スキャン・ライン２０で２つの青い辺８２および８４を追加する

（３）スキャン・ライン３５までレンダリングする

（４）スキャン・ライン３５で２つの赤い辺９２および９４を追加する

（５）最後までレンダリングする。

【００４０】同様に、図８の例によれば、辺情報１５には、下記が含まれることになる。

・辺８４は画素位置１００から始まり、辺８２は画素位置１００から始まる；

・辺９２は画素位置４０から始まり、辺９４は画素位置１６０から始まる；

・辺８４は７０スキャン・ラインだけ延び、辺８２は７０スキャン・ラインだけ延びる；

・辺８４は傾斜＝－１を有し、辺８４は傾斜＝＋１を有する；

・辺９２は傾斜＝０を有し、辺９４は傾斜＝０を有する；

・辺９２および９４は７０スキャン・ラインだけ延びる。

【００４１】上の命令ストリーム１４および辺情報１５の例とそれぞれが表現される形から、図８の画像７８では、画素位置（Ｘ）とスキャン・ライン値（Ｙ）によって、画像７８がレンダリングされる単一の出力空間が定義されることが認められる。しかし、本開示の原理を使用して、他の出力空間構成を実現することもできる。

【００４２】図８には、ラスタ画像画素データが含まれず、したがって、表示リスト１３の記憶部分１６には何も記憶する必要がない。この特徴については後で説明する。

【００４３】表示リスト・ストア１３は、画素シーケンシャル・レンダリング装置２０によって読み取られる。画素シーケンシャル・レンダリング装置２０は、通常は集積回路として実施される。画素シーケンシャル・レンダリング装置２０は、表示リストをラスタ画素のストリ

ームに変換し、このストリームは、たとえばプリンタ、ディスプレイまたはメモリ・ストアなどの別の装置に転送することができる。

【００４４】好ましい実施形態では、集積回路として画素シーケンシャル・レンダリング装置２０を説明するが、これは、ホスト・プロセッサなどの汎用処理ユニット上で実行可能な同等のソフトウェア・モジュールとして実施することができる。このソフトウェア・モジュールは、ディスク装置またはコンピュータ・ネットワークなどのコンピュータ可読媒体を介してユーザに配布することのできるコンピュータ・プログラム製品の一部を形成することができる。

【００４５】図３は、画素シーケンシャル・レンダリング装置２０、表示リスト・ストア１３および一時的レンダリング・ストア３０の構成を示す図である。画素シーケンシャル・レンダリング装置２０の処理ステージ２２には、命令実行機構３００、辺処理モジュール４００、優先順位決定モジュール５００、塗潰し色決定モジュール６００、画素合成モジュール７００および画素出力モジュール８００が含まれる。処理動作では、一時的ストア３０を使用するが、これは、上で述べたように表示リスト・ストア１３と同一の装置（たとえば磁気ディスクまたは半導体ＲＡＭ）を共用するか、速度最適化のために個々の格納部（ストア）として実施することができる。辺処理モジュール４００は、辺レコード・ストア３２を使用して、スキャン・ラインからスキャン・ラインへ順方向に運ばれる辺情報を保持する。優先順位決定モジュール５００は、優先順位特性および状況テーブル３４を使用して、各優先順位に関する情報と、スキャン・ラインがレンダリングされている間の辺交差に関する各優先順位の現在の状態を保持する。塗潰し色決定モジュール６００は、塗潰しデータ・テーブル３６を使用して、特定の位置で特定の優先順位の塗潰し色を決定するのに必要な情報を保持する。画素合成モジュール７００は、画素合成スタック３８を使用して、出力画素の値を決定するために複数の優先順位からの色が必要になる出力画素の、決定中の中間結果を保持する。

【００４６】表示リスト・ストア１３および上で詳細を示した他のストア３２ないし３８は、ＲＡＭ内で実施するか、他のデータ記憶技術で実施することができる。

【００４７】図３の実施形態に示された処理ステップは、処理パイプライン２２の形をとる。この場合、パイプラインのモジュールは、以下で説明する形でそれらの間でメッセージを受け渡しながら、画像データの異なる部分に対して並列に同時に実行することができる。もう１つの実施形態では、以下で説明するメッセージのそれぞれが、下流モジュールへの制御の同期転送の形をとることができ、上流処理は、下流モジュールがそのメッセージの処理を完了するまで中断される。

【００４８】命令実行機構３００は、命令ストリーム１４から命令を読み取り、処理し、その命令を、出力３９８を介してパイプライン２２内の他のモジュール４００、５００、６００および７００に転送されるメッセージにフォーマットする。好ましい実施形態では、命令ストリーム１４に、以下の命令を含めることができる。

【００４９】ＬＯＡＤ＿ＰＲＩＯＲＩＴＹ＿ＰＲＯＰＥＲＴＩＥＳ：　この命令は、優先順位特性および状況テーブル３４にロードされるデータと、そのデータがロードされるテーブル内のアドレスに関連する。命令実行機構３００は、この命令に出会った時に、優先順位特性および状況テーブル３４の指定された位置でのデータの記憶のためのメッセージを発行する。これは、このデータを含むメッセージをフォーマットし、処理パイプライン２２を介して、ストア動作を実行する優先順位決定モジュール５００に渡すことによって達成できる。

【００５０】ＬＯＡＤ＿ＦＩＬＬ＿ＤＡＴＡ：　この命令は、塗潰しデータ・テーブル３６にロードされるデータと、そのデータがロードされるテーブル内のアドレスに関連する。命令実行機構３００は、この命令に出会った時に、塗潰しデータ・テーブル３６の指定されたアドレスでのデータの記憶のためのメッセージを発行する。これは、このデータを含むメッセージをフォーマットし、処理パイプライン２２を介して、ストア動作を実行する塗潰しデータ決定モジュールに渡すことによって達成できる。

【００５１】ＬＯＡＤ＿ＮＥＷ＿ＥＤＧＥＳ＿ＡＮＤ＿ＲＥＮＤＥＲ：　この命令は、次のスキャン・ラインをレンダリングする時にレンダリング処理に導入される新しい辺１５の表示リスト・ストア１３内のアドレスに関連する。命令実行機構３００は、この命令に出会った時に、このデータを含むメッセージをフォーマットし、辺処理モジュール４００に渡す。辺処理モジュール４００は、新しい辺のアドレスを辺レコード・ストア３２に記憶する。指定されたアドレスにある辺は、次のスキャン・ラインをレンダリングする前に、最初のスキャン・ライン交差座標に基づいてソートされる。一実施形態では、辺は、表示リスト生成処理１２によってソートされる。別の実施形態では、辺は、画素シーケンシャル・レンダリング装置２０によってソートされる。

【００５２】ＳＥＴ＿ＳＣＡＮ　ＬＩＮＥ＿ＬＥＮＧＴＨ：　この命令は、レンダリングされるスキャン・ラインのそれぞれで作られる画素数に関連する。命令実行機構３００は、この命令に出会った時に、この値を辺処理モジュール４００および画素合成モジュール７００に渡す。

【００５３】ＳＥＴ＿ＯＰＡＣＩＴＹ＿ＭＯＤＥ：　この命令は、画素合成演算で不透明度チャネル（当技術分野ではアルファ・チャネルとも称する）を使用するかどうかを示すフラグに関連する。命令実行機構３００は、この命令に出会った時に、このフラグの値を画素合成モ

ジュール７００に渡す。

【００５４】命令実行機構３００は、通常は、命令をマッピングし、パイプライン動作に復号して、さまざまなモジュールに渡す、マイクロコードステートマシンによって形成される。或いは、その代わりに、対応するソフトウェア処理を使用することもできる。

【００５５】スキャン・ラインのレンダリング動作中の辺処理モジュール４００の動作を、図４を参照以下以下に説明する。スキャン・ラインのレンダリングのための初期条件は、以下の３つの辺レコードのストが使用可能であることである。これら３つのリストのいずれかまたはすべては空でもよい。これらのリストは、辺情報１５から取得されＬＯＡＤ＿ＮＥＷ＿ＥＤＧＥＳ＿ＡＮＤ＿ＲＥＮＤＥＲ命令によってセットされる新しい辺を含む新辺リスト４０２、前のスキャン・ラインから順方向に運ばれた辺レコードを含む主辺リスト４０４および、やはり前のスキャン・ラインから順方向に運ばれた辺レコードを含むスピル辺リスト４０６である。各辺レコードには、下記が含まれ得る。

【００５６】（ｉ）現在のスキャン・ライン交差座標（本明細書ではＸ座標と称する）

（ｉｉ）この辺の現在の線分が続くスキャン・ライン数のカウント（本明細書ではＮＹと称する。いくつかの＊施形態では、これをＹ限界と表現する場合がある）

（ｉｉｉ）各スキャン・ラインの後でこの辺レコードのＸ座標に加算される値（本明細書ではＤＸと称する）

（ｉｖ）各スキャン・ラインの後でこの辺レコードのＤＸに加算される値（本明細書ではＤＤＸと称する）

（ｖ）１つまたは複数の優先順位番号（Ｐ）

（ｖｉ）辺がスキャンラインと上向き（＋）に交差するか下向き（－）に交差するかを示す方向（ＤＩＲ）フラグ

（ｖｉｉ）リスト内の次の辺線分のアドレス（ＡＤＤ）。

【００５７】このようなフォーマットは、ベクトル、直交配置された辺および二次曲線に適合する。これ以外のパラメータ、たとえばＤＤＤＸの追加によって、このような配置が三次曲線に適合できるようになる。三次ベジェ・スプラインなど、いくつかの応用分野では、６階多項式（すなわちＤＤＤＤＤＤＸまで）が必要になる場合がある。

【００５８】図８の辺８４および９４の例では、スキャン・ライン２０での対応する辺レコードが、以下の表１に示されたものになる。

【００５９】

【表２】

TABLE I

Edge 84	Edge 92
X = 100	X = 40
NY = 70	NY = 70
DX = 1	DX = 0
DDX = 0	DDX = 0
P = 1	P = 0
DIR = (-)	DIR = (+)
ADD = (irrelevant in this example)	ADD = (irrelevant in this example)

【００６０】この説明では、レンダリング処理によって生成されつつあるスキャン・ラインに沿って画素から画素へステップする座標を、Ｘ座標と称し、スキャン・ラインからスキャン・ラインへとステップする座標を、Ｙ座標と称する。各辺リストには、メモリ内で連続的に配置された０個以上のレコードが含まれることが好ましい。ポインタ・チェインの使用を含む他の記憶配置も可能である。３つのリスト４０２、４０４および４０６のそれぞれのレコードは、スキャン・ライン交差（Ｘ）座標の順で配置される。これは、通常は、当初は辺入力モジュール４０８によって管理されるソート処理によって得られ、辺入力モジュール４０８は、辺情報を含むメッセージを命令実行機構３００から受け取る。各スキャン・ライン交差座標の整数部分だけを有意とみなすためにソートを緩和することが可能である。また、各スキャン・ライン交差座標を、現在のレンダリング処理によって作られる最小および最大のＸ座標にクランプされるとみなすことによってさらにソートを緩和することが可能である。適当な場合には、辺入力モジュール４０８は、メッセージを、出力４９８を介してパイプライン２２の下流のモジュール５００、６００および７００に受け渡す。

【００６１】辺入力モジュール４０８は、３つのリスト４０２、４０４および４０６のそれぞれへの参照を維持し、これらのリストから辺データを受け取る。これらの参照のそれぞれは、スキャン・ラインの処理の開始時に、各リスト内の最初の辺を参照するように初期設定される。その後、辺入力モジュール４０８は、選択されるレコードが３つの参照されるレコードからの最小のＸ座標を有するものになるように、３つの参照された辺レコードのうちの１つから辺レコードを選択する。複数のＸレコードが等しい場合には、それぞれが任意の順序で処理され、対応する辺交差が、下記の形で出力される。そのレコードの選択に使用された参照は、その後、そのリストの次のレコードに進められる。選択されたばかりの辺は、メッセージにフォーマットされ、辺更新モジュー

ル４１０に送られる。また、辺のいくつかのフィールド、具体的には現在のＸ、優先順位番号および方向フラグが、メッセージにフォーマットされ、このメッセージは、辺処理モジュール４００の出力４９８として優先順位決定モジュール５００に転送される。なお、本明細書に記載されたものより多数または少数のリストを使用する実施形態も可能である。

【００６２】辺を受け取った時に、辺更新モジュール４１０は、現在の線分が続くスキャン・ライン数のカウントをデクリメントする。そのカウントが０に達した場合には、次の線分アドレスによって示されるアドレスから新しい線分を読み取る。線分では、下記が指定される。

（ｉ）線分を読み取った直後に現在のＸ座標に加算される値

（ｉｉ）その辺の新しいＤＸ値

（ｉｉｉ）その辺の新しいＤＤＸ値

（ｉｖ）新しい線分が続くスキャン・ライン数の新しいカウント。

【００６３】示されたアドレスに使用可能な次の線分がない場合には、その辺に対してそれ以上の処理は実行されない。そうでない場合には、辺更新モジュール４１０は、その辺の次のスキャン・ラインのＸ座標を計算する。これには、通常は、現在のＸ座標をとり、これにＤＸ値を加算することが用いられる。ＤＸは、処理される辺の種類に応じて、必要であればＤＤＸ値を加算される場合がある。その後、辺は、複数の辺レコードの配列である辺プール４１２内の使用可能な空きスロットに書き込まれる。空きスロットがない場合には、辺更新モジュール４１０は、スロットが使用可能になるのを待つ。辺レコードが辺プール４１２に書き込まれたならば、辺更新モジュール４１０は、新しい辺が辺プール４１２に追加されたことを、信号線４１６を介して辺出力モジュール４１４に知らせる。

【００６４】スキャン・ラインのレンダリングのための初期条件として、辺出力モジュール４１４は、図４には図示されていないが、辺レコード３２内のリスト４０４および４０６に関連する次走辺リスト４２０および次スピル辺リスト４２２のそれぞれへの参照を有する。これらの参照のそれぞれは、当初は空のリスト４２０および４２２が構築される位置に初期設定される。辺が辺プール４１２に追加されたことを示す信号４１６を受け取った時に、辺出力モジュール４１４は、追加された辺が、次走辺リスト４２０に最後に書き込まれた辺（があれば）より小さいＸ座標を有するかどうかを判定する。これが真である場合には、順序付けの基準を侵害せずにその辺を走辺リスト４０４の末尾に追加することができないので、「スピル」が発生したという。スピルが発生した時には、その辺は、好ましくはソートされた次スピル辺リスト４２２を維持する形で、次スピル辺リスト４２２に挿入される。たとえば、これは、ソフトウェア・ソート・ルーチンを使用して達成することができる。いくつかの実施形態では、スピルが、極端に大きいＸ座標など、他の条件によってトリガされる場合がある。

【００６５】辺プール４１２に追加された辺が、次走辺リスト４２０に最後に書き込まれた辺（があれば）以上のＸ座標を有し、辺プール４１２に使用可能な空きスロットがない場合には、辺出力モジュール４１４は、辺プール４１２から、最小のＸ座標を有する辺を選択し、その辺を次走辺リスト４２０の末尾に追加し、この処理で次走辺リスト４２０を延長する。辺プール４１２内の、その辺によって占められていたスロットは、空きとしてマークされる。

【００６６】辺入力モジュール４０８は、これら３つの入力リスト４０２、４０４および４０６のすべてからすべての辺を読み取り、転送した後に、スキャン・ラインの終りに達したことを示すメッセージをフォーマットし、そのメッセージを、優先順位決定モジュール５００と辺更新モジュール４１０の両方に送る。そのメッセージを受け取った時に、辺更新モジュール４１０は、それが現在実行しているすべての処理が完了するのを待ち、その後、このメッセージを辺出力モジュール４１４に転送する。そのメッセージを受け取った時に、辺出力モジュール４１４は、辺プール４１２からの残りの辺レコードのすべてを、Ｘ順で次の走辺リスト４０４に書き込む。その後、次走辺リスト４２０および走辺リスト４０４への参照を、辺入力モジュール４０８と辺出力モジュール４１４の間で交換し、同様の交換を、次スピル辺リスト４２２とスピル辺リスト４０６に関しても実行する。この形で、次のスキャン・ラインのための初期条件が確立される。

【００６７】辺レコードの挿入時に次スピル辺リスト４２２をソートするのではなく、そのような辺レコードを、単にリスト４２２の末尾に追加することができ、スキャン・ラインの末尾で、現在のスピル・リスト４０６との交換の前にソートされるリスト４２２は、次のスキャン・ラインの辺ラスタ化でアクティブになる。エッジをソートする他の方法では、より少数またはより多数のリストを使用することができ、また、異なるソート・アルゴリズムを使用することができる。

【００６８】上記から、辺交差メッセージが、スキャン・ライン順および画素順（すなわち、まずＹでソートされ、次にＸでソートされる）で優先順位決定モジュール５００に送られること、および各辺交差メッセージに、それに適用される優先順位のラベルが付けられることを推論することができる。

【００６９】図１２Ａは、辺の線分が受け取られる時に辺処理モジュール４００によって作成される可能性があるアクティブ辺レコード４１８の具体的な構造を示す図である。辺の最初の線分がステップ（直交）線分である場合には、辺のＸ値を、最初の線分の「Ｘステップ」と

称する変数に加算して、アクティブ化された辺のＸ位置を得る。そうでない場合には、辺のＸ値を使用する。これは、新しい辺レコードの辺が、Ｘ₁，，，＋Ｘ，，，によってソートされなければならないことを意味する。したがって、最初の線分のＸ₁，，，は、辺のソートを単純化するために０でなければならない。最初の線分のＹ値は、アクティブ辺レコード４１８のＮＹフィールドにロードされる。アクティブな辺のＤＸフィールドは、ベクトルまたは二次線分のＤＸフィールド離別子からコピーされ、ステップ線分の場合には０がセットされる。図１２Ａに示されたｕフラグは、線分が上向き（図１３Ａに関連する説明を参照されたい）である場合にセットされる。ｑフラグは、線分が二次線分の場合にセットされ、そうでない場合にはクリアされる。ｉフラグが設けられ、これは、線分が不可視である場合にセットされる。ｄフラグは、辺が、関連するクリッピング・レベルなしに直接クリッピング・オブジェクトして使用され、閉じた曲線に適用可能である時にセットされる。線分の実際の優先順位レベルまたはレベル・アドレスは、新しい辺レコードの対応するフィールドからアクティブ辺レコード４１８のレベル（ＡＤＤＲ）フィールド（Level(Addr)）にコピーされる。アクティブ辺レコード４１８の線分アドレス／ＤＤＸフィールド（Seg.Addr(DDX)）は、線分リスト内の次の線分のアドレスであるか、線分が二次線分の場合にはセグメントのＤＤＸ値からコピーされるかのいずれかである。線分アドレスは、辺レコードを終了させるのに使用される。その結果、好ましい実施形態では、二次曲線のすべて（すなわち、ＤＤＸフィールドを使用する曲線）が、辺レコードの終端線分になる。

【００７０】図１２Ａから、他のデータ構造も可能であり、たとえばより高次の多項式の実装が使用される場合にはそれが必要であることを記解されたい。さらに、線分アドレスおよびＤＤＸフィールドは、異なるフィールドに分離することができ、代替実施態様に合わせて追加のフラグを設けることができる。

【００７１】図１２Ｂは、辺処理モジュール４００で使用される、上で説明した好ましい実施形態の辺レコードの配置を示す図である。辺プール４１２は、新アクティブ辺レコード４２８、現アクティブ辺レコード４３０およびスピル・アクティブ辺レコード４３２によって補足される。図１２Ｂからわかるように、レコード４０２、４０４、４０６、４２０および４２２は、ある時点でレングリングされる辺の数に応じて、サイズを動的に変更することができる。各レコードには、新辺リスト４０２の場合にはＬＯＡＤ＿ＥＤＧＥＳ＿ＡＮＤ＿ＲＥＮＤＥＲ命令に組み込まれたＳＩＺＥ値によって決定される、限界値が含まれる。このような命令に出会った時には、ＳＩＺＥを検査し、０でない場合には、新しい辺レコードのアドレスをロードし、リスト４０２の限界サイズを決定する限界値を計算する。

【００７２】好ましい実施形態では、辺レコードの処理のために配列とそれに関連するポインタを使用するが、たとえばリンク・リストなどの、他の実施態様を使用することができる。これらの他の実施態様は、ハードウェア・ベース、ソフトウェア・ベースまたはその組合せとすることができる。

【００７３】図８に示された画像７８の具体的なレングリングを、図１０に示されたスキャン・ライン３４、３５および３６に関連してこれから説明する。この例では、次のスキャン・ラインの新しいＸ座標の計算が、説明を明瞭にするために省略され、図１２Ｃないし図１２Ｉでは、出力辺交差が、辺プール４１２のレジスタ４２８、４３０および４３２の１つから導出される。

【００７４】図１２Ｃは、スキャン・ライン３４（半透明の青い三角形８０の最上部）のレングリングの終りでの、上で述べたリストの状態を示す図である。スキャン・ライン３４には、新しい辺がなく、したがって、リスト４０２が空であることに留意されたい。主辺リスト４０４および次主辺リスト４２０のそれぞれには、辺８２および８４だけが含まれる。リストのそれぞれには、対応するポインタ４３４、４３６および４４０が含まれ、これらは、スキャン・ライン３４の完了時に、対応するリスト内の次の空きレコードを指す。各リストには、対応するリストの末尾を指すために必要な、アスタリスク（＊）によって示される限界ポインタ４５０も含まれる。リンク・リストを使用する場合には、リンク・リストに、対応する機能を実行するヌル・ポインタ終端子が含まれるので、このような限界ポインタは不要になる。

【００７５】上で述べたように、各スキャン・ラインの始めに、次主辺リスト４２０と主辺リスト４０４が交換され、新しい辺が、新辺リスト４０２に受け取られる。残りのリストはクリアされ、ポインタのそれぞれは、各リストの最初のメンバを指すようにセットされる。スキャン・ライン３５の始めでは、配置は図１２Ｄのようになる。図１２Ｄからわかるように、レコードには、図１０から辺９２、９４、８４および８２に対応することがわかる４つのアクティブな辺が含まれる。

【００７６】図１２Ｅを参照すると、レングリングが開始される時に、新辺レコード４０２の最初の線分が、アクティブ辺レコード４２８にロードされ、主辺リスト４０４およびスピル辺リスト４０６の最初のアクティブな辺レコードが、それぞれレコード４３０および４３２にコピーされる。この例では、スピル辺リスト４０６は空であり、したがって、ローディングは行われない。レコード４２８、４３０および４３２内の辺のＸ位置が比較され、辺交差が、最小のＸ位置を有する辺について発行される。この場合、発行される辺は、辺９２に対応する辺であり、この辺がその優先順位値と共に出力される。その後、ポインタ４３４、４３６および４３８が、リスト内の次のレコードを指すように更新される。

【００７７】その後、辺交差が発行される辺が更新され（この場合、その位置にＤＸ＝０を加算することによって）、辺プール４１２にバッファリングされ、この辺プール４１２は、この例では３つの辺レコードを保持するサイズになる。発行された辺が現れたリスト（この場合ではリスト４０２）内の次の項目が、対応するレコード（この場合ではレコード４２８）にロードされる。これを図１２Ｆに示す。

【００７８】さらに、図１２Ｆから明らかなとおり、レジスタ４２８、４３０および４３２の間の比較によって、もう一度最小のＸ値を有する辺が選択され、適切な次の辺交差（Ｘ＝８５、Ｐ＝２）として出力される。そして、同様に、選択され出力された辺は、更新され、辺プール４１２に追加され、適当なポインタのすべてがポインクリメントされる。この場合、更新される値は、Ｘ←Ｘ＋ＤＸによって与えられ、ここでは、８４＝８５－１として与えられる。また、図からわかるように、新しい辺のポインタ４３４が、この場合では新辺リスト４０２の末尾に移動される。

【００７９】図１２Ｇでは、最小の現行Ｘ値を有すると識別された次の辺が、やはり、レジスタ４３０から取得され、辺交差（Ｘ＝１１５、Ｐ＝２）として出力される。そして、辺の更新が発生し、図示のように、その値が辺プール４１２に追加される。この時、辺プール４１２は満杯になり、ここから、最小のＸ値を有する辺が選択され、出力リスト４２０に発行され、対応する限界ポインタが、それ相応に移動される。

【００８０】図１２Ｈからわかるように、次の最小の辺交差は、レジスタ４２８からのものであり、これが出力される（Ｘ＝１６０、Ｐ＝１）。やはり辺プール４１２が更新され、次に小さいＸ値が出力リスト４２０に発行される。

【００８１】スキャン・ライン３５の終りに、図１２Ｉからわかるように、Ｘ値の小さい順で、辺プール４１２の内容が出力リスト４２０にフラッシュされる。図１２Ｊからわかるように、次主辺リスト４２０と主辺リスト４０４は、次のスキャン・ライン３６のレンダリングに備えて、ポインタを交換することによって交換される。交換の後に、図１２Ｊからわかるように、主辺リスト４０４の内容には、スキャン・ライン３６上の現在の辺のすべてが含まれ、これらはＸ位置の順で配置され、これによって、高速のレンダリングを容易にする便利なアクセスが可能になる。

【００８２】通常、新しい辺は、Ｘ位置の昇順で辺処理モジュール４００によって受け取られる。新しい辺が現れる時には、その位置が更新され（次にレンダリングされるスキャン・ラインのために計算され）、これによって、以下の処置が決定される。

【００８３】（ａ）更新された位置が信号線４９８に出力された最後のＸ位置より小さい場合には、新しい辺は、主スピル・リスト４０６へのソートされる挿入であり、対応する限界レジスタが更新される。

【００８４】（ｂ）そうでない場合に、空間があるならば、その辺は辺プール４１２内で保存される。

【００８５】前述から明白なとおり、辺プール４１２は、ラスタ化画像の次のスキャン・ラインのレンダリングに備えた順序付きの形でのリストの更新を助ける。さらに、辺プール４１２のサイズは、多数の順序付けられていない辺に適合するために変更することができる。しかし、実際には、辺プール４１２が、一般にグラフィック処理システムの処理速度および使用可能なメモリに依存する、実用的な限界を有することは明白である。制限的な意味では、辺プール４１２を省略することができるが、これは、通常は、更新された辺を、次出力辺リスト４２０へのソートされた挿入にすることを必要とする。しかし、好ましい実施形態では、上で述べたスピル・リストの使用を介する通常の出来事として、この状況が回避される。スピル・リストを設けることによって、好ましい実施形態を、実用的なサイズの辺プールを用いて実施でき、なおかつ、ソフトウェア集中的なソート手順に頼らずに比較的複雑な辺交差を処理することができる。辺プールとスピル・リストが辺交差の複雑さに適合するのに不十分になる場合では、これは稀な場合であるが、ソート法を使用することができる。

【００８６】スピル・リスト手順が使用される場合の例を、図１４Ａに示す。図１４Ａでは、３つの任意の辺６０、６１および６３が、スキャン・ラインＡおよびＢの間の相対位置で任意の辺６２と交差する。さらに、スキャン・ラインＡおよびＢのそれぞれについて実際に表示される画素位置６４が、スパン画素位置ＣないしＪとして図示されている。辺プール４１２が３つの辺レコードを保存するサイズである、上で説明した例では、このような配置だけでは、図１４Ａに示された隣接するスキャン・ラインの間で発生する３つの辺の交差に適合するのに不十分であることは明白である。

【００８７】図１４Ｂに、スキャン・ライン上の辺６０、６１および６３をレンダリングした後の辺レコードの状態を示す。辺交差Ｉは、最も最近に発行された辺交差であり、辺プール４１２は、次のスキャン・ラインであるスキャン・ラインＢのための、それぞれ辺６０、６１および６３の更新された Ｘ値Ｅ、Ｇおよび Ｉで満杯になっている。辺６２は、現アクティブ辺レコード４３０にロードされ、辺プール４１２が満杯なので、辺６０に対応する最小のＸ値が、出力辺リスト４２０に出力される。

【００８８】図１４Ｃでは、次の辺交差が発行され（辺６２のＸ＝Ｊ）、対応する更新された値、この場合はスキャン・ラインＢのＸ＝Ｃが決定される。新たに更新された値Ｘ＝Ｃは、出力リスト４２０からコピーされた最も最近の値Ｘ＝Ｅより小さいので、現在の辺レコードと

それに対応する新たに更新された値が、出力スピル・リスト４２２に直接に転送される。

【００８９】図１４Ｄに、スキャン・ラインＢの開始時の辺レコードの状態を示す。この図では、主リストおよび出力リストと、それらに対応するスピル構成要素が交換されていることがわかる。最初に発行される辺を決定するために、辺６０を現アクティブ辺レジスタ４３０にロードし、辺６２を、スピル・アクティブ辺レジスタ４３２にロードする。Ｘ値が比較され、最小のＸ値（Ｘ＝Ｃ）を有する辺６２が発行され、更新され、辺プール４１２にロードされる。

【００９０】辺の発行と更新は、主辺リスト４０４内の残りの辺について継続され、スキャン・ラインの終りに、辺プール４１２がフラッシュされて、図１４Ｅに示された状況になる。図１４Ｅからは、辺６０ないし６３のそれぞれが、次のスキャン・ラインでのレンダリングのために適当に順序付けられ、スキャン・ラインＢ上で正しく発行され、レンダリングされたことがわかる。

【００９１】上記から明らかになるように、スピル・リストは、複雑な辺交差状況の存在のもとで、辺ラスタ化順序の維持をもたらす。さらに、このリストがサイズにおいて動的に変更可能であることによって、辺交差の数と複雑さの大きい変化を、例外的に複雑な辺交差以外のすべての辺交差でソート手順に積る必要なしに処理できる。

【００９２】好ましい実施形態では、辺プール４１２は、８つの辺レコードを保存するサイズにされ、リスト４０４および４２０のサイズは、それらに関連するスピル・リスト４０６および４２２と一緒に、動的に変更可能であり、これによって、複雑な辺交差要件を有する大きい画像を処理するために十分な範囲が提供される。

【００９３】優先順位決定モジュール５００の動作を、図５を参照して以下に説明する。辺処理モジュール４００からの着信メッセージ９８は、優先順位データ設定メッセージ、塗潰しデータ設定メッセージ、辺交差メッセージおよびスキャン・ラインの終りメッセージが含まれる可能性があるが、優先順位更新モジュール５０６によって読み取られる前に、まず先入れ先出し（ＦＩＦＯ）バッファ５１８を通過する。ＦＩＦＯ５１８は、辺処理モジュール４００の動作と優先順位決定モジュール５００の動作を分離するように働く。優先順位状態テーブル５０２は、上で述べたテーブル３４０一部を含むが、各オブジェクトの優先順位に関する情報を保持するのに使用される。ＦＩＦＯ５１８は、エッジ交差のスキャン・ライン全体の辺処理モジュール４００からの受取りと優先順位状態テーブル５０２への転送を単一の処置で可能にするサイズであることが好ましい。これによって、優先順位決定モジュール５００が、同一の画素（Ｘ）位置での複数の辺交差を効率的に処理できるようになる。優先順位状態テーブル５０２の各レコードには、以下が記録される。

【００９４】（ｉ）この優先順位が、奇数－偶数塗潰し規則または非ゼロ・ワインディング塗潰し規則の適用によって決定される内部／外部状態を有するかどうかを示す塗潰し規則フラグ

（ｉｉ）この優先順位をもたらす辺が交差するたびに塗潰し規則によって示される形で変更される塗潰しカウント

（ｉｉｉ）この優先順位がクリッピングと塗潰しのどちらに使用されるかを示すクリッパ・フラグ

（ｉｖ）クリッパ・フラグがセットされている辺について、クリッピング・タイプが「クリップ・イン」と「クリップ・アウト」のどちらであるかを記録するクリップ・タイプ・フラグ

（ｖ）この優先順位をもたらすクリップ・イン・タイプのクリップ領域に入る時にデクリメントされ、出る時にインクリメントされ、この優先順位をもたらすクリップ・アウト・タイプのクリップ領域に入る時にインクリメントされ、出る時にデクリメントされる、クリップ・カウント

（ｖｉ）「ニード・ビロウ」フラグと称する、この優先順位がそれ未満のレベルを最初に計算することを必要とするかどうかを記録するフラグ。

【００９５】クリッピング・オブジェクトは、当技術分野で既知であり、特定の新しいオブジェクトを表示するように働くのではなく、画像内の別のオブジェクトの形状を変更するように働く。クリッピング・オブジェクトは、さまざまな視覚的効果を達成するために、オンにし、オフにすることもできる。たとえば、図８のオブジェクト８０は、オブジェクト９０に対して、オブジェクト９０のうちでクリッピング・オブジェクト８０の下にある部分を除去するように働くクリッピング・オブジェクトとして構成することができる。これは、クリッピング境界内における、オブジェクト９０の下にある、クリッピングされていなければオブジェクト９０の不透明さによって隠されるオブジェクトまたは画像を見せるという効果を有する。

【００９６】図１３Ａおよび図１３Ｂに、奇数－偶数規則と非ゼロ・ワインディング規則の応用例を示す。非ゼロ・ワインディング規則の目的のために、図１３Ａに、オブジェクト７０の辺７１および７２が、辺が下向きと上向きのどちらであるかに従って概念上の向きをどのように割り振られるかを示す。閉じた境界を形成するために、辺は、境界に沿って始点と終点が重なる形でリンクする。塗潰し規則（後で適用され、説明される）の目的のために辺に与えられる向きは、線分が定義される順序と独立である。辺の線分は、レンダリングの方向に対応する、追跡される順序で定義される。

【００９７】図１３Ｂに、２つの下向きの辺７３および７６と、３つの上向きの辺７４、７５および７７を有す

る単一のオブジェクト（五線星形）を示す。奇数－偶数規則は、各辺が対象のスキャン・ラインと交差するたびにブール値を単純にトグルし、したがって、効果的にオブジェクト色をオンにするかオフにすることによって動作する。非ゼロ・ワインディング規則では、交差する辺の向きに応じて塗債しカウント値をインクリメントまたはデクリメントする。図１３Ｂでは、このスキャン・ラインで出会う最初の２つの辺７３および７６が下向きであり、したがって、これらの辺の横断によって、塗債しカウントがインクリメントされ、それぞれ＋１および＋２になる。このスキャン・ラインが出会う次の２つの辺７４および７７は、上向きであり、したがって、塗債しカウントがそれぞれ＋１および０にデクリメントされる。

【００９８】いくつかの実施形態では、この情報の一部が、表示リスト１３および前に説明したさまざまな辺リストの辺に関連し、辺交差メッセージの一部として優先順位決定モジュール５００に転送される。具体的に言うと、塗債し規則フラグ、クリッパ・フラグ、クリップ・タイプ・フラグおよびニード・ピロウ・フラグを、この形で処理することができる。

【００９９】図５に戻って、優先順位更新モジュール５０６は、それが処理を完了したものまでのスキャン・ライン交差座標を記録するカウンタ５２４を維持する。これを、優先順位更新モジュール５０６の現行Ｘと称する。スキャン・ラインの開始時の初期値は０である。

【０１００】ＦＩＦＯ５１８の先頭で受け取った辺交差メッセージを検査する際に、優先順位更新モジュール５０６は、辺交差メッセージのＸ交差値とその現行Ｘを比較する。辺交差メッセージのＸ交差値が、優先順位更新モジュール５０６の現行Ｘ以下の場合には、優先順位更新モジュール５０６は、その辺交差メッセージを処理する。辺交差メッセージの処理は、２つの形態になり、「通常辺処理」（下で説明する）は、辺交差メッセージの最初の優先順位によって示される優先順位状態テーブル５０２内のレコードが、クリップ優先順位でないことを示すクリッパ・フラグを有する時に使用され、そうでない場合には、「クリップ辺処理」（下で説明する）が実行される。

【０１０１】優先順位がある画素でアクティブになるのは、その画素が、その優先順位の塗債し規則に従って優先順位に適用される境界辺の内部にあり、その優先順位のクリップ・カウントが０の場合である。優先順位は、それが最も上のアクティブな優先順位である場合、または、それより上位のアクティブな優先順位のすべてが、対応するニード・ピロウ・フラグをセットしている場合に、公開される。この形で、画素値を、公開された優先順位の塗債しデータだけを使用して生成することができる。

【０１０２】ある優先順位のニード・ピロウ・フラグは、表示リストの情報内で確立され、そのフラグがセットされていなければ、問題の優先順位の下のアクティブな優先順位のすべてが、レンダリング中の画素値に寄与しないことを画素生成システムに知らせるのに使用される。このフラグは、最終的な画素値に全く寄与しないはずの余分な合成演算を防ぐのに適当な場合にクリアされる。

【０１０３】「通常辺処理」には、辺交差メッセージ内の優先順位のそれぞれについて、その優先順位によって示される優先順位状態テーブル・レコードのフィールドに関して、以下のステップが含まれる。

【０１０４】（ｉ）現在の優先順位の現在の塗債しカウントを記録し、（ｉｉ）現在の優先順位の塗債し規則が、（ａ）奇数－偶数である場合には、塗債しカウントが現在０以外であれば塗債しカウントに０をセットし、そうでない場合には０以外の値をセットし、（ｂ）非ゼロ・ワインディングである場合には、塗債しカウントをインクリメントまたはデクリメント（辺の方向フラグに応じて）し、（ｉｉｉ）新しい塗債しカウントと記録された塗債しカウントを比較し、一方が０で他方が０以外の場合には、現在の優先順位に対して「アクティブ・フラグ更新」（下で説明する）動作を実行する。

【０１０５】いくつかの実施形態では、辺交差メッセージのそれぞれに複数の優先順位を置くのではなく、優先順位ごとに別々の辺交差メッセージを使用することができる。

【０１０６】アクティブ・フラグ更新動作には、まず現在の優先順位のための新しいアクティブ・フラグを確立することが含まれる。アクティブ・フラグは、優先順位状態テーブル５０２のその優先順位の塗債しカウントが０以外であり、その優先順位のクリップ・カウントが０の場合に０以外になり、そうでない場合には、アクティブ・フラグは０になる。アクティブ・フラグ更新動作の第２ステップは、アクティブ・フラグ配列５０８内の現在の優先順位によって示される位置に、決定されたアクティブ・フラグを格納し、現在の優先順位の優先順位状態テーブルのニード・ピロウ・フラグが０の場合には、不透明アクティブ・フラグ配列５１０の現在の優先順位によって示される位置にもアクティブ・フラグを格納することである。

【０１０７】「クリップ辺処理」には、辺交差メッセージの最初の優先順位によって示される優先順位状態テーブル・レコードのフィールドに関して、以下のステップが含まれる。

【０１０８】（ｉ）現在の優先順位の現在の塗債しカウントを記録し、（ｉｉ）現在の優先順位の塗債し規則が、（ａ）奇数－偶数である場合には、塗債しカウントが現在０以外であれば塗債しカウントに０をセットし、そうでない場合には０以外の値をセットし、（ｂ）非ゼロ・ワインディングである場合には、塗債しカウントを

インクリメントまたはデクリメント（辺の方向フラグに応じて）し、（ｉｉｉ）新しい塗潰しカウントと記録された塗潰しカウントを比較し、（ａ）新しい塗潰しカウントが０であり、記録された塗潰しカウントが０である場合、または、新しい塗潰しカウントが０以外であり、記録された塗潰しカウントが０以外である場合には、０、（ｂ）現在の優先順位のクリップ・タイプ・フラグがクリップ・アウトであり、記録された塗潰しカウントが０であり、新しい塗潰しカウントが０以外である場合、または、現在の優先順位のクリップ・タイプ・フラグがクリップ・インであり、記録された塗潰しカウントが０以外であり、新しい塗潰しカウントが０である場合には、＋１、（ｃ）それ以外の場合には、－１のクリップ・デルタ値を決定し、（ｉｖ）辺交差メッセージの最初の優先順位の後のすべての後続優先順位について、その後続優先順位によって示される優先順位状態テーブル内のレコードのクリップ・カウントに決定されたクリップ・デルタ値を加算し、その処理で、クリップ・カウントが、０以外から０になった場合または０から０以外になった場合に、その後続優先順位に対して上で説明したアクティブ・フラグ更新動作を実行する。各クリップ・カウントの初期値は、前に説明したＬＯＡＤ＿ＬＥＶＥＬ＿ＰＲＯＰＥＲＴＩＥＳ命令によってセットされることに留意されたい。クリップ・カウントは、通常は、各優先順位に影響するクリップ・イン優先順位の数に初期設定される。

【０１０９】いくつかの実施形態では、優先順位がクリップに関連付けられず、その代わりに、辺交差メッセージで与えられるすべての優先順位のクリップ・カウントが直接にインクリメントまたはデクリメントされる。この技法は、たとえば、クリップ形状が単純であり、複雑な塗潰し規則の適用が不要な時に使用することができる。この特定の応用例では、辺によって制御されるレベルのクリップ・カウントは、上向きの辺の場合にインクリメントされ、下向きの辺の場合にデクリメントされる。反時計回りに記述された単純な閉じた曲線は、クリップ・インとして働き、時計回りに記述された単純な閉じた曲線は、クリップ・アウトとして働く。

【０１１０】辺交差メッセージのＸ交差値が、優先順位更新モジュール５０６の現行Ｘを超える時には、優先順位更新モジュール５０６は、生成する画素数のカウントすなわち、辺交差メッセージのＸ交差値と現行Ｘの間の差を形成し、このカウントを優先順位生成メッセージにフォーマットし、接続５２０を介して優先順位生成モジュール５１６に送る。その後、優先順位更新モジュール５０６は、所与の個数の画素の処理が完了したことを示す優先順位生成モジュール５１６からの信号５２２を待つ。信号５２２を受け取った時に、優先順位更新モジュール５０６は、その現行Ｘに、辺交差メッセージのＸ交差値をセットし、上で説明した処理を継続する。

【０１１１】優先順位生成モジュール５１６は、テーブル３４内でも形成される、各優先順位に関する情報を保持するのに使用される、優先順位データ・テーブル５０４に関して動作する。優先順位データ・テーブル５０４の各レコードには、以下が含まれる可能性がある。

【０１１２】（ｉ）塗潰しテーブル・アドレス
（ｉｉ）塗潰しタイプ
（ｉｉｉ）ラスタ演算コード
（ｉｖ）アルファ・チャネル演算コード
（ｖ）「ソース・ポップ」フラグ
（ｖｉ）「デスティネーション・ポップ」フラグ
（ｖｉｉ）本明細書で「ｘ独立」フラグと称する、この優先順位の色が所与のＹに対して一定であるかどうかを記録するフラグ。

【０１１３】優先順位生成メッセージの受取り５２０の際に、優先順位生成モジュール５１６は、供給されたカウントによって示される回数だけ「画素優先順位生成動作」（下で説明する）を実行し、その後すべてに、動作を完了したことを知らせる信号５２２を優先順位更新モジュール５０６に送る。

【０１１４】各画素の優先順位生成動作には、まず不透明アクティブ・フラグ配列５１０に対して優先順位エンコーダ５１４（たとえば、４０９６対１２ビット優先順位エンコーダ）を使用して、最高の不透明アクティブ・フラグの優先順位の数を決定することが含まれる。この優先順位（もし存在すれば）は、優先順位データ・テーブル５０４のインデクシングに使用され、そのように参照されたレコードの内容は、優先順位生成モジュール５１６から塗潰し優先順位メッセージ出力５９８に形成され、塗潰し色決定モジュール６００に送られる。さらに、優先順位が前のステップによって決定された（すなわち、少なくとも１つの不透明アクティブ・フラグがセットされた）場合には、決定された優先順位が保持され、これを「現行優先順位」と称する。優先順位が決定されなかった場合には、現行優先順位に０をセットする。その後、優先順位生成モジュール５１６は、アクティブ・フラグ配列５０８に対して変更済み優先順位エンコーダ５１２を繰り返し使用して、現行優先順位を超える最小のアクティブ・フラグを決定する。そのように決定された優先順位（があれば）は、優先順位データ・テーブル５０４のインデクシングに使用され、そのように参照されたレコードの内容は、塗潰し優先順位メッセージに形成され、塗潰し色決定モジュール６００に送られ５９８、その後、決定された優先順位が、現行優先順位の更新に使用される。このステップは、優先順位が決定されなくなる（すなわち、現行優先順位を超える、アクティブ・フラグでフラグを立てられた優先順位がなくなる）まで、繰り返し使用される。その後、優先順位生成モジュール５１６は、画素の終りメッセージを形成し、塗潰し色決定モジュール６００に送る。

【０１１５】上で説明した基本動作に対する好ましい特徴として、優先順位生成モジュール５１６は、シーケンスの最初の画素を処理する間に、塗潰し色決定モジュール６００に転送する各メッセージのｘ独立フラグの値を記録する。転送されるメッセージのすべてでｘ独立フラグが指定されている場合には、隣接する辺交差の間の画素のスパンのすべての後続メッセージを、カウント－１の単一の反復指定によって置換することができる。これは、反復メッセージを作り、このシーケンスのその後の処理のすべての代わりに塗潰し色決定モジュール６００に送ることによって行われる。

【０１１６】上で説明した基本動作に対するもう１つの好ましい特徴として、優先順位生成モジュール５１６は、各レベル生成メッセージの後に、接続５２２を介して優先順位更新モジュール５０６に最高の不透明優先順位を送る。優先順位更新モジュール５０６は、これをストア５２６に保持する。その後、優先順位更新モジュール５０６は、メッセージのＸ交差が現行Ｘより大きいことを単純にテストするのではなく、画素優先順位生成メッセージを作る前に、メッセージのＸ交差が現行Ｘより大きいこと、およびメッセージ内のレベルのうちの少なくとも１つが、最高の不透明優先順位以上であることをテストする。これを行うことによって、より少ない数の画素優先順位決定動作を実行することができ、より長い反復シーケンスを生成することができる。

【０１１７】いくつかの実装で所望されるように、動作の反復メッセージまたはシーケンスが使用されない場合、同様の機能を、優先順位生成モジュール５１６の出力にキャッシュまたはＦＩＦＯ（図示せず）を組み込むことを介して達成できる。これは、たとえば４セル・キャッシュによって実施できる。キャッシュを用いると、優先順位更新モジュール５０６が、キャッシュがロードされると同時に作業を継続でき、これによって、出力５９８へのキャッシュのフラッシュと独立に次の優先順位レベルを生成できるようになる。

【０１１８】図８および図９Ａ，Ｂに示されたグラフィック・オブジェクトの例を使用して、上で説明した優先順位更新処理を、図１２Ｃないし図１２Ｊおよび図１５Ａないし図１５Ｅに示された辺の交差を使用して、スキャン・ライン３５に関して示すことができる。

【０１１９】図１５Ａないし図１５Ｅは、優先順位テーブル５０２および５０４の動作を示す図であり、優先順位テーブル５０２および５０４は、好ましい実施形態では、配列５０８および５１０とエンコーダ５１２および５１４と共に、レベル・アクティブ化テーブル５３０と称する単一のテーブルに合併される。図１５Ａからわかるように、辺交差メッセージは、辺処理モジュール４００からスキャン・ラインの順で受け取られ、テーブル５３０にロードされ、テーブル５３０は、優先順位の順で配置される。辺交差メッセージには、この例では、辺横断の非ゼロ・ワインディング規則に従うインクリメント方向が含まれる。優先順位テーブル５３０内の項目をセットしないことも可能である。

【０１２０】図示のレベル・アクティブ化テーブル５３０には奇数－偶数規則に従って辺から決定される塗潰しカウントのための列項目が含まれる。ニード・ビロウ・フラグは、優先順位の特性であり、ＬＯＡＤ＿ＰＲＩＯＲＩＴＩＥＳ＿ＰＲＯＰＥＲＴＩＥＳ命令の一部としてセットされる。ニード・ビロウは、テーブル５３０がロードされる時に、すべての優先順位レベルについてセットされる。「クリップ・カウント」および「塗潰しインデックス・テーブル」などの他の列を使用することができるが、この例では、説明を簡単にするために省略する。どのレベルもアクティブでない場合、対応する項目に０がセットされる。さらに、配列５１０および５０８の値は、後続の辺交差を受け取った後に、テーブル５３０から更新される。

【０１２１】図１５Ａから、便宜上、複数のレコードが明瞭さのために省略されていることは明白である。通常、レベル・アクティブ化テーブル５３０には、優先順位の順で配置された、以下のレコードが含まれるはずである。

【０１２２】・塗潰しカウント
・クリップ・カウント
・塗潰しタイプ
・下記を含むアクティブ化条件およびフラグ
（ｉ）ニード・ビロウ・フラグ
（ｉｉ）クリップ・タイプ
（ｉｉｉ）クリッパ・フラグ
・下記を含む合成用グラフィック演算およびフラグ
（ｉ）ラスタ演算コード
（ｉｉ）アルファ・チャネル演算コード
（ｉｉｉ）「ソース・ポップ」フラグ
（ｉｖ）「デスティネーション・ポップ」フラグ
（ｖ）ｘ独立フラグ
・塗潰し規則
・属性
・塗潰しテーブル・インデックス。

【０１２３】テーブル５３０の内容は、優先順位決定モジュール５００で使用されない場合に、画素生成用の塗潰し色決定モジュール６００と合成演算用の画素合成モジュール７００のそれぞれにメッセージとして渡される。

【０１２４】スキャン・ライン３５の最初の辺交差（図１２Ｅ）を図１５Ａに示すが、図１５Ａでは、Ｐ＝１の場合に、塗潰しカウントが、非ゼロ・ワインディング規則に従って辺の値に更新される。下のオブジェクトは存在しないので、「ニード・ビロウ」は、０にセットされるレベルである。

【０１２５】テーブル５３０の前の状態はセットされていないので、配列５１０および５０８は、セットされないままになり、優先順位エンコーダ５１４は、優先順位の出力をディスエーブルされる。これは、優先順位生成モジュール５１６によって解釈され、優先順位生成モジュール５１６は、スキャン・ライン３５の最初の空白の部分である「オブジェクトなし」優先順位（たとえばＰ＝０）のカウントｎ＝４０（画素）を出力する。

【０１２６】図１５Ｂは、図１２Ｆの辺交差を受け取った時の配置を示す図である。塗潰しカウントが更新される。その後、配列５１０および５０８は、テーブル５３０からの前の最高レベルを用いてセットされる。この時点で、モジュール５１６は、半透明の三角形８０との交差の前の不透明な赤いオブジェクト９０の辺９６を表す、カウントｎ＝４５、Ｐ＝１を出力する。

【０１２７】図１５Ｃは、図１２Ｇの辺交差を受け取った時の配置を示す図である。非ゼロ・ワインディング規則のゆえに、塗潰しカウントが下向きに調節されていることに留意されたい。現在の辺交差を受け取る前に有効であるオブジェクトは、不透明ではないので、変更済み優先順位エンコーダ５１２が、ｎ＝（１１５－８５）＝３０画素の現行として出力される最高のアクティブ・レベルとして優先順位Ｐ＝２を選択するのに使用される。

【０１２８】図１５Ｄは、図１２Ｈの辺交差を受け取った時の配置を示す図である。前に変更されたＰ＝２の「ニード・ビロウ」が、アクティブ配列５０８に転送され、したがって、優先順位エンコーダが、ｎ＝（１６０－１１５）＝４５画素の現行である値Ｐ＝１を出力することができることに留意されたい。

【０１２９】図１５Ｅは、図１２Ｉの辺交差を受け取った時の結果を示す図であり、ｎ＝（１８０－１６０）＝２０画素のＰ＝０の出力が供給される。

【０１３０】したがって、優先順位モジュール５００は、スキャン・ラインのすべての画素に関する、画素のカウントと、対応する優先順位表示値を出力する。

【０１３１】塗潰し色決定モジュール６００の動作を、図６を参照して、以下に説明する。優先順位判定モジュール５００からの着信メッセージ５９８は、塗潰しデータ設定メッセージ、反復メッセージ、塗潰し優先順位メッセージ、画素の終りメッセージおよびスキャン・ラインの終りメッセージを含み、まず、塗潰しルックアップおよび制御モジュール６０４に渡される。塗潰しルックアップおよび制御モジュール６０４は、塗潰し色決定モジュール６００のさまざまな構成要素による使用のために、現行Ｘ位置カウンタ６１４および現行Ｙ位置カウンタ６１６を維持する。

【０１３２】スキャン・ラインの終りメッセージを受け取った時には、塗潰しルックアップおよび制御モジュール６０４は、現行Ｘ位置カウンタ６１４を０にリセットし、現行Ｙ位置カウンタ６１６をインクリメントする。スキャン・ラインの終りメッセージは、その後、画素合成モジュール７００に渡される。

【０１３３】塗潰しデータ設定メッセージを受け取った時には、塗潰しルックアップおよび制御モジュール６０４は、塗潰しデータ・テーブル３６の指定された位置６０２にそのデータを記憶する。

【０１３４】反復メッセージを受け取った時には、塗潰しルックアップおよび制御モジュール６０４は、反復メッセージからのカウントだけ現行Ｘ位置カウンタ６１４をインクリメントする。反復メッセージは、その後、画素合成モジュール７００に渡される。

【０１３５】画素の終りメッセージを受け取った時には、塗潰しルックアップおよび制御モジュール６０４は、やはり現行Ｘ位置カウンタ６１４をインクリメントし、この画素の終りメッセージは、その後、画素合成モジュール７００に渡される。

【０１３６】塗潰し優先順位メッセージを受け取った時には、塗潰しルックアップおよび制御モジュール６０４は、下記の処理を含む動作を実行する。

【０１３７】（ｉ）塗潰し優先順位メッセージからの塗潰しタイプを使用して、テーブル３６のレコード・サイズを選択する。

【０１３８】（ｉｉ）塗潰し優先順位メッセージからの塗潰しテーブル・アドレスと、上で決定されたレコード・サイズを使用して、塗潰しデータ・テーブル３６からレコードを選択する。

【０１３９】（ｉｉｉ）塗潰し優先順位メッセージからの塗潰しタイプを使用して、塗潰し色の生成を実行するサブモジュールを決定し、選択する。サブモジュールには、ラスタ画像モジュール６０６、フラット・カラー・モジュール６０８、リニア・ランプ・カラー・モジュール６１０および不透明タイル・モジュール６１２を含めることができる。

【０１４０】（ｉｖ）決定されたレコードを、選択されたサブモジュール６０６ないし６１２に供給する。

【０１４１】（ｖ）選択されたサブモジュール６０６ないし６１２が、供給されたデータを使用して、色と不透明度の値を決定する。

【０１４２】（ｖｉ）決定された色と不透明度は、塗潰し色メッセージからの残りの情報、すなわち、ラスタ演算コード、アルファ・チャネル演算コード、ソース・ポップ・フラグおよびデスティネーション・ポップ・フラグと組み合わされて、色合成メッセージを形成し、この色合成メッセージは、接続６９８を介して画素合成モジュール７００に送られる。

【０１４３】好ましい実施形態では、決定された色と不透明度が、８ビットの精度の赤、緑、青および不透明度の４つ組であり、通常の形で３２ビット／画素が与えられる。しかし、シアン、マゼンタ、黄および黒の、不透明度を含有する４つ組か、他の多数の既知の色表現のう

ちの１つを、その代わりに使用することができる。赤、緑、青および不透明度の場合を、下の説明で使用するが、この説明は、他の場合にも適用することができる。

【０１４４】ラスタ画像モジュール６０６、フラット・カラー・モジュール６０８、リニア・ランプ・カラー・モジュール６１０および不透明タイル・モジュール６１２の動作を、これから説明する。

【０１４５】フラット・カラー・モジュール６０８は、供給されたレコードを、３つの８ビットの色成分（通常は赤、緑および青の成分と解釈される）と１つの８ビットの不透明度値（通常は、指定された色によって覆われる画素の割合の尺度と解釈され、０は覆われないすなわち完全な透明を意味し、２５５は完全に覆われるすなわち完全な不透明を意味する）を含む固定フォーマットのレコードと解釈する。この色と不透明度の値は、接続６９８を介して直接に出力され、それ以上の処理なしで、＊決定された色および不透明度を形成する。

$$\mathrm{clamp}(x)=\begin{cases}255 & 255\leq x\\ \lfloor x\rfloor & 0\leq x<255\\ 0 & x<0\end{cases}$$

【０１５０】代替実装では、リニア・ランプ・カラー・モジュール６１０は、供給されたレコードを、３つの色成分および１つの不透明度成分に関連する、４組の３つの定数ｃｘ、ｃｙおよびｄを含む固定フォーマットのレコードと解釈する。これらの４組のそれぞれについて、結果の値ｒは、次式を使用して、３つの定数を現行Ｘカウントであるｘおよび現行Ｙカウントであるｙと組み合わせることによって計算される。

【０１５１】

ｒ＝ｃｌａｍｐ（ｃｘ×ｘ＋ｃｙ×ｙ＋ｄ）

ここで、関数ｃｌａｍｐは、上で定義されたものである。

【０１５２】そのように作られた４つの結果は、色と不透明度の値に形成される。この色と不透明度の値は、接続６９８を介して直接に出力され、それ以上の処理なしで決定された色および不透明度を形成する。

【０１５３】同一の結果を与える他の算術計算を使用することができる。

【０１５４】不透明タイル・モジュール６１２は、供給されたレコードを、３つの８ビット色成分、１つの８ビット不透明度値、整数のＸ位相（ｐｘ）、Ｙ位相（ｐｙ）、Ｘスケール（ｓｘ）、Ｙスケール（ｓｙ）および６４ビットのマスクを含む固定フォーマットのレコードと解釈する。これらの値は、表示リスト生成から発し、通常は、元のページ記述に含まれる。ビット・マスク内のビット・アドレスａは、次の式で決定される。

【０１５５】ａ＝（（ｘ／２’’＋ｐｘ）ｍｏｄ　８）＋（（ｙ／２’’＋ｐｙ）ｍｏｄ　８）×８　。

【０１５６】ビット・マスク内のアドレス「ａ」のビットが検査される。検査されるビットが１の場合には、レコードからの色と不透明度が、モジュール６１２の出力に直接にコピーされ、決定された色および不透明度を形成する。検査されるビットが０の場合には、３つの０成分値を有する色と０の不透明度値が形成され、決定された色および不透明度として出力される。

【０１４６】リニア・ランプ・カラー・モジュール６１０は、供給されたレコードを、３つの色成分および１つの不透明度成分に関連する４組の定数ｃｘ、ｃｙおよびｄと、リニア・ランプの基準点の座標である２つの位置値ｘｂａｓｅおよびｙｂａｓｅを含む固定フォーマットのレコードと解釈する。基準点では、色成分と不透明度成分が、それに関連付けられたｄ値を有する。

【０１４７】４組のそれぞれについて、結果の値ｒは、次式を使用して、現在のＸＹ座標とｘｂａｓｅおよびｙｂａｓｅ定数に３つの定数を組み合わせることによって計算される。

【０１４８】ｒ＝ｃｌａｍｐ（ｃｘ×（Ｘ－ｘｂａｓｅ）＋ｃｙ×（Ｙ－ｙｂａｓｅ）＋ｄ）

ここで、関数ｃｌａｍｐは、次のように定義される。

【０１４９】

$$\mathrm{clamp}(x)=\begin{cases}255 & 255\leq x\\ \lfloor x\rfloor & 0\leq x<255\\ 0 & x<0\end{cases}$$

。

【０１５７】ラスタ画像モジュール６０６は、供給されたレコードを、６つの定数ａ、ｂ、ｃ、ｄ、ｘｂａｓｅおよびｙｂａｓｅと、サンプリングされるラスタ画像画素データ１６の各ラスタ・ライン内のビット数の整数カウント（ｂｐｌ）と、画素タイプとを含む固定フォーマットのレコードと解釈する。画素タイプは、ラスタ画像画素データ内の画素データ１６を、次のうちのどれとして解釈するかを示す。

【０１５８】（ｉ）１ビット毎画素の白黒不透明画素
（ｉｉ）１ビット毎画素の不透明黒または透明の画素
（ｉｉｉ）８ビット毎画素のグレイ・スケール不透明画素
（ｉｖ）８ビット毎画素の黒不透明スケール画素
（ｖ）２４ビット毎画素の不透明３色成分画素
（ｖｉ）３２ビット毎画素の３色成分と不透明度の画素
他の多くのフォーマットが可能である。

【０１５９】ラスタ画像モジュール６０６は、画素タイプ・インジケータを使用して、ビット単位の画素サイズ（ｂｐｐ）を決定する。その後、ラスタ画像画素データ１６内のビット・アドレスａを、次式から計算する。

【０１６０】ａ＝ｂｐｐ＊⌊ａ×（ｘ－ｘｂａｓｅ）＋ｃ×（ｙ－ｙｂａｓｅ）⌋＋ｂｐｌ×⌊ｂ×（ｘ－ｘｂａｓｅ）＋ｄ×（ｙ－ｙｂａｓｅ）⌋　。

【０１６１】レコードＢ６０２からの画素タイプに従って解釈された画素は、ラスタ画像画素データ１６内の計算されたアドレス「ａ」から取り出される。この画素は、３つの８ビット色成分と１つの８ビット不透明度成分を

有するために、必要に応じて展開される。「展開」とは、たとえば、８ビット／画素のグレイ・スケール不透明ラスタ画像からの画素が、赤、緑および青の成分のそれぞれに適用されるサンプリングされた８ビット値と、完全な不透明がセットされた不透明度成分を持つことを意味する。これが、画素合成モジュール７００への決定された色および不透明度の出力６９８を形成する。

【０１６２】その結果、表示可能オブジェクト内で有効なラスタ画素データは、メモリ内の画像画素データ１６へのマッピングの決定を介して得られる。これによって、ラスタ画素データのオブジェクト・ベース画像へのアフィン変換が効率的に実施され、これは、グラフィック・オブジェクトとの合成が発生する可能性がある場合に画像ソースからの画素データをフレーム・ストアに転送する従来技術の方法より効率的である。

【０１６３】上記に対する好ましい特徴として、任意選択として、ラスタ画像画素データ１６内の画素間の補間を、まず中間結果ｐおよびｑを次式に従って計算することによって実行することができる。

【０１６４】

$$p = a \times (x - x_{base}) + c \times (y - y_{base})$$

$$q = b \times (x - x_{base}) + d \times (y - y_{base})$$

。

【０１６５】次に、ラスタ画像画素データ１６内の４画素のビット・アドレスa_{ss}、a_{sl}、a_{ls}およびa_{ll}を、次式に従って決定する。

$$a_{ss} = \lfloor bpp \times \lfloor p \rfloor \rfloor + \lfloor bpl \times \lfloor q \rfloor \rfloor$$

$$a_{sl} = a_{ss} + bpp$$

$$a_{ls} = a_{ss} + bpl$$

$$a_{ll} = a_{ss} + bpl + bpp$$

。

【０１６６】次に、結果の画素成分値ｒを、次式に従って、色成分および不透明度成分のそれぞれについて決定する。

$$r = interp(interp(get(a_{ss}), get(a_{sl}), p), interp(get(a_{ls}), get(a_{ll}), p), q)$$

ここで、関数ｉｎｔｅｒｐは、次のように定義される。

$$interp(a, b, c) = a + (b - a) \times (c - \lfloor c \rfloor)$$

。

【０１６７】上の諸式で、表現└値┘＝ｆｌｏｏｒ（値）であり、ｆｌｏｏｒ演算では、値の端数部分の切捨が行われる。

【０１６８】ｇｅｔ関数は、所与のビット・アドレスでの、ラスタ画像画素データ１６からサンプリングされた現行画素成分の値を返す。いくつかの画像タイプのいくつかの成分について、これが暗黙の値になる可能性があることに留意されたい。

【０１６９】上記に対する好ましい特徴として、任意選択として、供給されたレコードから読み取られるタイル・サイズを用いる剰余演算によって現行Ｘ位置カウンタ６１４および現行Ｙ位置カウンタ６１６から導出されるｘおよびｙの値を上式で使用することによって、画像タイリングを実行することができる。

【０１７０】多数の他のこのような強調し色生成サブモジュールが可能である。

【０１７１】画素合成モジュール７００の動作をこれから説明する。強調し色決定モジュール６００からの着信メッセージには、反復メッセージ、色合成メッセージ、画素の終りメッセージおよびスキャン・ラインの終りメッセージが含まれるが、このメッセージは、順番に処理される。

【０１７２】反復メッセージまたはスキャン・ラインの終りメッセージを受け取った時には、画素合成モジュール７００は、それ以上の処理なしで画素出力ＦＩＦＯ７０２にそのメッセージを転送する。

【０１７３】色合成メッセージを受け取った時には、画素合成モジュール７００は、概要を示せば、色合成メッセージからの色と不透明度を、色合成メッセージからのラスタ演算およびアルファ・チャネル演算に従って、画素合成スタック３８からポップされた色および不透明度と組み合わせる。その後、結果を画素合成スタック３８にプッシュする。色合成メッセージの受取り時に実行される処理の説明は、下で与える。

【０１７４】画素の終りメッセージを受け取った時には、画素合成モジュール７００は、画素合成スタック３８から色と不透明度をポップする。ただし、スタック３８が空の場合には不透明の白の値が使用される。結果の色と不透明度は、画素出力ＦＩＦＯに転送される画素出力メッセージに形成される。

【０１７５】既知の合成アプローチは、ポーター（Ｐｏｒｔｅｒ，Ｔ）およびダフ（Ｄｕｆｆ，Ｔ）著「Ｃｏｍｐｏｓｉｔｉｎｇ Ｄｉｇｉｔａｌ Ｉｍａｇｅｓ」，Ｃｏｍｐｕｔｅｒ Ｇｒａｐｈｉｃｓ誌，Ｖｏｌ．１８ Ｎｏ．３，１９８４年，第２５３～２５９ページに記載されている。ポーターダフ合成演算の例を、図２１に示す。しかし、このようなアプローチでは、合成によって形成される交差領域内のソースおよびデスティネーションの色の処理だけが可能であり、その結果、交差領域内の透明度の影響に適合できないという点で、不完全である。その結果、ポーターおよびダフによって定義されたラスタ演算は、透明度の存在下で基本的に動作不能である。

【０１７６】色合成メッセージを受け取った時に画素合成モジュール７００によって実行される処理を、これから説明する。

【０１７７】色合成メッセージを受け取った時に、画素合成モジュール７００は、まず、ソースの色と不透明度を形成する。これは、色合成メッセージでポップ・ソース・フラグがセットされていない限り、色合成メッセージで供給された色と不透明度が使用され、ポップ・ソー

ス・フラグがセットされている場合には、その代わりに画素合成スタック３８から色がポップされる。この時、すなわち、画素合成スタックのポップ中に、画素合成スタック３８が空であることがわかった場合、エラー表示なしで不透明の白の色値が使用される。次に、デスティネーションの色と不透明度が、画素合成スタック３８からポップされる。ただし、色合成メッセージでデスティネーション・ポップ・フラグがセットされていない場合には、スタック・ポインタが「ポップ」動作中に変更されず、その結果スタック３８の最上部からの読み取りになる。

【０１７８】ソースの色および不透明度をデスティネーションの色および不透明度と組み合わせる方法を、図７Ａないし図７Ｃを参照して以下に説明する。色および不透明度の値は、通常は０から２５５までの範囲の８ビット値として記憶されるが、ここでは説明の目的のために、０から１までの範囲（すなわち正規化されている）とみなす。２画素を合成する目的のために、各画素が２つの領域、すなわち、完全に不透明な領域と完全に透明な領域に分割されるものとみなし、不透明度値は、これら２つの領域の比率の表示であるとみなされる。図７Ａに、図示されない３成分色値と、不透明度値（ｓｏ）を有するソース画素７０２を示す。ソース画素７０２の影付きの領域は、画素７０２の完全に不透明な部分７０４を表す。同様に、図７Ａの影付きでない領域は、ソース画素７０２のうちで、完全に透明であるとみなされる部分７０６を表す。図７Ｂに、ある不透明度値（ｄｏ）を＊有するデスティネーション画素７１０を示す。デスティネーション画素７１０の影付きの領域は、画素７１０の完全に不透明な部分７１２を表す。同様に、画素７１０は、完全に透明な部分７１４を有する。ソース画素７０２およびデスティネーション画素７１０の不透明領域は、組合せのために、互いに直交するとみなされる。これら２つの画素のオーバーレイ７１６を、図７Ｃに示す。対象の３つの領域が存在し、これには、ｓｏ×（１－ｄｏ）の面積を有するデスティネーション外ソース７１８、面積ｓｏ×ｄｏを有するソース・デスティネーション交差７２０および、（１－ｓｏ）×ｄｏの面積を有するソース外デスティネーション７２２が含まれる。これら３つの領域のそれぞれの色値は、概念上は独立に計算される。デスティネーション外ソース領域７１８は、その色をソース色から直接にとる。ソース外デスティネーション領域７２２は、その色をデスティネーション色から直接にとる。ソース・デスティネーション交差領域７２０は、その色をソース色とデスティネーション色の組合せからとる。上で説明した他の動作とは別個の、ソース色とデスティネーション色の組合せの処理は、ラスタ演算と称され、これは、画素合成メッセージからのラスタ演算コードによって指定される、１組の関数のうちの１つである。好ましい実施形態に含まれるラスタ演算の一部を、表２に示す。

【０１７９】

【表３】

TABLE 2

Raster operation code	Operation	Comment
0x00	r = 0	BLACKNESS
0x01	r = src & dest	SRCAND
0x02	r = src & ~dest	SRCERASE
0x03	r = src	SRCCOPY
0x04	r = ~src & dest	
0x05	r = dest	NOP
0x06	r = src ^ dest	SRCINVERT
0x07	r = src | dest	SRCPAINT
0x08	r = ~(src | dest)	NOTSRCERASE
0x09	r = ~(src ^ dest)	
0x0a	r = ~dest	DSTINVERT
0x0b	r = src | ~dest	
0x0c	r = ~src	NOTSRCCOPY
0x0d	r = ~src | dest	MERGEPAINT
0x0e	r = ~(src & dest)	
0x0f	r = 0xff	WHITENESS
0x10	r = min(src, dest)	
0x11	r = max(src, dest)	
0x12	r = clamp(src + dest)	
0x13	r = src	
0x14	r = clamp(src - dest)	
0x15	r = dest	
0x16	r = clamp(dest - src)	
0x17	r = clamp(src + dest) where dest is signed	
0x18	r = threshold (dest, src)	
0x19	r = threshold (src, dest)	
0x1a	r = ~ dest	
0x1b	o = luminance (dest, src)	
0x1c	r = ~src	
0x1d	o = okey(dest; src +/- o)	

【０１８０】各関数は、ソース色およびデスティネーション色の対応する色成分の対のそれぞれに適用されて、結果の色において同様の成分が得られる。多くの他の関数が可能である。

【０１８１】画素合成メッセージからのアルファ・チャネル演算も考慮されている。アルファ・チャネル演算は、３つのフラグを使用して実行され、フラグのそれぞれは、ソース画素７０２とデスティネーション画素７１０のオーバーレイ７１６内の対象の領域のうちの１つに対応する。領域のそれぞれについて、領域の不透明度値が形成され、この不透明度値は、アルファ・チャネル演算で対応するフラグがセットされていない場合には０、それ以外の場合にはその領域の面積である。

【０１８２】結果の不透明度は、領域の不透明度の合計から形成される。結果の色の各成分は、領域の色と領域の不透明度の対のそれぞれの積の和を結果の不透明度で除算することによって形成される。

【０１８３】結果の色および不透明度は、画素合成スタック３８にプッシュされる。

【０１８４】エクスプレッションツリーは、画像の形成に必要な合成演算を記述するのにしばしば使用され、通常は、葉ノード、単項ノードおよび２項ノードを含む複数のノードが含まれる。葉ノードは、エクスプレッションツリーの最も外側のノードであり、子ノードを持たず、画像のプリミティブ要素を表す。単項ノードは、ツリーの単項演算子の下の部分から来る画素データを変更する演算を表す。２項ノードは、通常は、左と右のサブツリーに分岐し、各サブツリー自体は、少なくとも１つの葉ノードを含むエクスプレッションツリーである。

【０１８５】任意の形状のオブジェクトとの合成の時には、上で述べたさまざまなスタック演算が、画像の異なる区域について異なり、特定の位置でアクティブなオブジェクトに依存するという問題が生じる。

【０１８６】図１７Ａおよび図１７Ｂに、ソース（Ｓ）とデスティネーション（Ｄ）の間の通常の２項演算（エクスプレッションツリーとして図示）を示す。実際に実行される演算とは無関係に、図１７Ａの２項演算は、下に示すアクティビティの４つの場合または領域に分解される。

【０１８７】１．（Ａ）Ｓがアクティブ、（Ｂ）Ｄが非

アクティブ
２．（Ａ）Ｓが アクティブ、（Ｂ）Ｄが アクティブ
３．（Ａ）Ｓが 非アクティブ、（Ｂ）Ｄが アクティブ
４．（Ａ）Ｓが 非アクティブ、（Ｂ）Ｄが 非アクティブ。

【０１８８】ケース４は、必ず無演算（ＮＯＰ）が実行される必要があることをもたらし、その結果、２進木のアクティブ・レベルには３つの異なる組合せが存在することになる。この概念の３項、４項およびさらに高次の木への拡張は、当業者には明白である。

【０１８９】その結果、合成スタック３８（２項の例の場合）を構築する時に、上で識別した３つの演算のうちの１つを、スタックによって実施する必要がある。さらに、スタック内の各オブジェクトに関連する異なる演算は、レベル・アクティブ化テーブルでそのオブジェクトの下にあるものに依存する。ペインタのアルゴリズムで発生する、単純なＯＶＥＲ演算を使用するオブジェクトのレンダリングの場合に、これは問題にならない。しかし、他の合成演算に関して、スタック演算は、合成演算のオペランドのアクティビティに依存して変更される必要がある。これは、スタック演算を提供するレベルをクリッピングすることによって行うことができるが、各レベルに適用されるクリップの数が、急激に増加し、スタック演算の処理が困難になる。問題になる演算の例が、あるオブジェクト（オペランド）が、他方のオブジェクトをクリッピング（その境界を変更）し、交差領域で可変透明度を有する場合の、図２１に示されたポーターーダフ演算のＯＵＴおよびＲＯＵＴである。

【０１９０】この問題に対処するために、本明細書で「アクティビティ」テーブルと称するもう１つのテーブルを設ける。このテーブルは、レベル・アクティブ化テーブルの付属品として働いて、上で述べた代替処置の論理的決定をもたらす。

【０１９１】図１８Ａに、基本的に３つの部分を含む、包括的なアクティビティ・テーブル８００を示す。第１部分８０２は、処理中の特定の塗潰しレベルのアクティブ化条件を提供する。第２部分８０４には、それぞれのレベル（具体的には２項の例の）に適当な、上で参照した異なる処置のそれぞれが含まれる。第３部分８０６は、ソース・オブジェクトまたはデスティネーション・オブジェクトが特定のレベルでアクティブであるかどうかを示す。処置部分８０４に含まれる項目は、具体的な演算そのものとするか、その代わりに、適当な場合にレベル・アクティブ化テーブルへのポインタとすることができることに留意されたい。

【０１９２】また、さまざまな演算によって、他の演算にデータを供給することができる場合と、できない場合があることに留意されたい。その結果、アクティビティ・テーブル８００は、演算がデータを供給するさまざまな条件を示すフラグを組み込むように変更することができる。

【０１９３】アクティビティ・テーブルの好ましい形態のデータ構造を、図１８Ｂにテーブル８１０として示す。テーブル８１０には、そのデータを使用する次の演算の項目へのポインタ８１４（Ｎｅｘｔ＿Ｌｅｖｅｌ）と、その演算がデータを提供するエクスプレッションツリーの枝（Ｓｒｃ＿Ａｃｔｉｖｅ／Ｄｅｓｔ＿ａｃｔｉｖｅ）を示すフラグ８０６（または、３項以上のツリーが使用される場合にはフラグの組）が含まれる。テーブル８１０には、その演算がソース枝またはデスティネーション枝にデータを供給するかどうかを示すフラグ８１６も含まれる。そうである場合には、次のレベルの項目のＳｒｃ＿ａｃｔｉｖｅまたはＤｅｓｔ＿Ａｃｔｉｖｅフラグ８０６は、この演算のアクティビティ状態８１８が変化した時に、それ相応に調節される。演算は、特定の組合せにおいてのみデータを供給するので、これを示すために、別のフラグ８１２（ｄａｔａ＿ｉｎ＿＊）が設けられる。フラグ８１２とＳｒｃ／Ｄｅｓｔ＿Ａｃｔｉｖｅフラグ８０６の組合せによって、あるレベルのアクティビティ状態が決定される。さらに、どの演算も、それ自体のアクティビティ状態が変化した場合には次のレベルの状態を変更するだけでよいので、ノード・アクティブ・フラグ８１８が、そのような状況を監視するために設けられる。

【０１９４】したがって、右側の親ノードについて、Ｐｕｓｈ動作と、次の演算レコードのＤｅｓｔ＿ａｃｔｉｖｅフラグをアクティブ化することが必要である。左側の親ノードについて、デスティネーションがすでにアクティブである可能性があることに留意して、辺交差のＳｒｃ＿ａｃｔｉｖｅフラグをアクティブ化することが必要である。

【０１９５】図１８Ｂでは、アクティブ化条件８０２に、葉ノードのアクティブ化を決定する塗潰し規則と、レベル・アクティブ化テーブルについて上で説明した形で使用される塗潰しカウントが含まれる。クリップ・カウントも、上で説明した形で動作する。辺の交差によって、テーブル８１０の（ソース）レベルがアクティブ化される。

【０１９６】レベルのアクティブ化状態が変化した時には、その変化が、Ｎｅｘｔ＿Ｌｅｖｅｌ項目８１４によって指されるレベルに伝播される。Ｄａｔａ＿ｉｓ＿Ｓｒｃフラグ８１６の状態に応じて、Ｓｒｃ＿Ａｃｔｉｖｅ／Ｄｅｓｔ＿Ａｃｔｉｖｅフラグ８０６が、次のレベルで変更される。この変更は、次のレベルの状態も変化する場合に伝播される。テーブル項目には、それぞれ場合１、２および３の演算が格納される。これは、レベル・アクティブ化テーブル内のレベルへのポインタとするか、実際の演算（たとえば、アルファ演算、カラー演算、塗潰しタイプおよびスタック演算フラグ）とすることができる。その代わりに、演算が必要でない場合に

は、これらをＮＵＬＬにすることができる。

【０１９７】あるノード・レベルのアクティブ化状態は、Ｓ∩Ｄ￣ｏｐ、Ｓ∩Ｄｏｐ、Ｓ￣∩Ｄｏｐのそれぞれのｄａｔａ＿ｉｎフラグ群８１２と、そのノード・レベルのＳｒｃ／Ｄｅｓｔ＿ａｃｔｉｖｅフラグ８０６によって決まる、エクスプレッションツリーの次の演算のためのデータがあるかどうかによって決定される。これは、このテーブルではＮｏｄｅ＿ＡｃｔｉｖｅフラグＳ１８として図示されている。

【０１９８】この実装の具体的な例を、図１９に示されるエクスプレッションツリー８３０について、図２０Ａに示される、対応する初期アクティビティ・テーブル８２０を用いて示す。

【０１９９】エクスプレッションツリー８３０は、オペランドＡ、ＢおよびＣのページへのレンダリングを提供し、後者は、図示の完全さのために、ツリー８３０では右の葉ノード、ＰＡＧＥとして示されている。ＰＡＧＥは、常にアクティブであり、画像出力空間全体を含み、したがって、アクティビティ・テーブル８２０から省略することができる。

【０２００】ＢとＣは葉ノードなので、これらは、テーブル８２０の低位レベルを形成し、それぞれが、それぞれの演算子の合成スタックへのプッシュを引き起こすことのできるアクティブ化演算８０４をもたらす。これらのそれぞれが右側の葉ノードなので、まずＣがプッシュされ、オペランドＣに対する演算がないので、Ｓ￣∩ＤｏｐはＮＯＰになる。ｄａｔａ＿ｉｎ＿＊ｏｐフラグ８１２、Ｎｅｘｔ＿Ｌｅｖｅｌフラグ８１４およびＤａｔａ＿ｉｓ＿ＳｒｃフラグＳ１６も更新される。オペランドＢは、対応する処置をもたらす。

【０２０１】アクティビティ・テーブル８２０の次のレベルは、左の葉ノードＡとそれに対応する演算子Ｏｐ２によって形成される。このレベルのアクティブ化演算８０４は、それぞれが演算の開の差を表す修飾子ａまたはｂによってそれぞれが変更されるＡｏｐ２ＢであるＳ￣∩ＤｏｐおよびＳ∩Ｄｏｐのそれぞれを用いて更新される。演算Ｏｐ２は、Ｓのデータを提供するだけであり、これは、Ｄがアクティブで Ｓ がアクティブでない（すなわちＳ￣∩Ｄｏｐ）場合にスタックからＢをポップするアクティビティによって表される。

【０２０２】テーブル８２０の次のレベルは、Ｏｐ１に関し、アクティブ化演算８０４でのそれぞれ修飾された結果ａ、ｂおよびｃを作る。最後のレベルｏｖｅｒでは、ＰＡＧＥが常にアクティブなので、Ｓ∩Ｄ￣ｏｐとＳ￣∩Ｄｏｐは、スタックにプッシュされるＮＯＰをもたらす。単純な交差Ｓ∩Ｄｏｐ内だけで、ｏｖｅｒがアクティブになる。

【０２０３】この例について、Ａ、ＢおよびＣが、当初は非アクティブであり、これらのオペランドが、その後、順番にアクティブ化される場合に何が起きるかを検討する。

【０２０４】Ａが最初にアクティブ化される場合、ＡＯｐ２ａがこのレベルでアクティブ化され、Ｓｒｃ＿ＡｃｔｉｖｅフラグＳ０６の設定に反映される。Ｓ∩Ｄ￣フラグＳ１２がセットされているので（Ｂがまだアクティブではないので）、Ｎｏｄｅ＿ＡｃｔｉｖｅフラグＳ１８が、このレベルについてセットされる。Ｎｏｄｅ＿Ａｃｔｉｖｅの状態が変化したので、Ｏｐ１レベルのＳｒｃ＿ＡｃｔｉｖｅフラグＳ０６がセットされる。Ｄａｔａ＿ｉｓ＿Ｓｒｃ８１６は、ＡＯｐ２レベルでセットされることに留意されたい。Ｏｐ１レベルは、Ｓｒｃ＿Ａｃｔｉｖｅと（Ｄｅｓｔ＿Ａｃｔｉｖｅ） ￣ を有するので（Ｃがまだアクティブではないので）、Ｏｐ１ａが、このレベルでアクティブ化される。Ｓ∩Ｄ￣がセットされているので、Ｎｏｄｅ＿Ａｃｔｉｖｅ８１８が、Ｏｐ１レベルについてセットされる。Ｎｏｄｅ＿Ａｃｔｉｖｅ８１８の状態が変化したので、ｏｖｅｒレベルのＳｒｃ＿ＡｃｔｉｖｅフラグＳ０６がセットされる。ｏｖｅｒレベルはＳｒｃ＿ａｃｔｉｖｅとＤｅｓｔ＿Ａｃｔｉｖｅを有する（ＰＡＧＥが常にアクティブなので）ので、ｏｖｅｒはこのレベルでアクティブ化される。Ｓ∩Ｄがセットされているので、Ｎｏｄｅ＿Ａｃｔｉｖｅがセットされ、Ｎｏｄｅ＿Ａｃｔｉｖｅの状態は変化しない。その後、これ以上の処置は不要である。合成スタック３８は、この段階で、テーブル８２０から確立でき、図２０Ｂに示されたものになる。

【０２０５】図２０Ｃに移って、Ｂが次にアクティブ化される場合、Ｓ∩Ｄ￣がセットされている（いずれにせよＤは関係ない）ので、このレベルでＰｕｓｈ Ｂがアクティブ化される。Ｎｏｄｅ＿Ａｃｔｉｖｅ８１８は、このレベルについてセットされる。Ｎｏｄｅ＿Ａｃｔｉｖｅの状態が変化したので、ＡＯｐ２レベルのＤｅｓｔ＿Ａｃｔｉｖｅフラグがセットされる。その後、ＡＯｐ２ｂＢがアクティブ化され、ＡＯｐ２ａＢが非アクティブ化される。Ｓ∩Ｄがセットされているので、Ｎｏｄｅ＿Ａｃｔｉｖｅはセットされたままになり、Ｎｏｄｅ＿Ａｃｔｉｖｅの状態は、ＡＯｐ２ａに関しては変化しない。これ以上の処置は発生しない。合成スタック３８は、図２０Ｄに示されたものになる。

【０２０６】図２０Ｅからわかるように、Ｃが次にアクティブ化される場合、このレベルでＰｕｓｈ Ｃがアクティブ化される。Ｓはアクティブであり、Ｄは無関係なので、Ｎｏｄｅ＿Ａｃｔｉｖｅ８１８は、このレベルについてセットされる。Ｎｏｄｅ＿ａｃｔｉｖｅが変化したので、Ｏｐ１レベルの Ｄｅｓｔ＿Ａｃｔｉｖｅフラグがセットされる。Ｏｐ１ｂがアクティブ化されるので、Ｏｐ１ａは非アクティブ化される。Ｓ∩Ｄのデータがセットされているので、Ｎｏｄｅ＿Ａｃｔｉｖｅはセットされたままになる。Ｏｐ１レベルではｎｏｄｅ＿Ａｃｔｉｖｅが変化しないので、これ以上の処置は不要であ

る。合成スタック３８は、図２０Ｆに示されたものになる。

【０２０７】この手順は、図１９のエクスプレッションツリー全体の評価について継続され、したがって、さまざまなアクティブ化条件８０２およびアクティビティ・インジケータ８０４による要求に応じて合成スタックにプッシュすることのできるさまざまな演算が確立される形で確立されるアクティビティ・テーブル８２０をもたらす。この形で、クリッピングまたは実行される他の演算の種類に無関係に、正しいレベルの正しい演算である状態で、評価されるエクスプレッションツリーの複雑さと無関係に、スタックを維持することができる。重要なことに、エクスプレッションツリーの大部分が、表示リストの形成を介して評価されるが、表示リストの生成は、通常は、クリッピングなどのさまざまなオブジェクトの演算の変化を説明することができず、これらの演算は、合成式の評価中に実施することが必要になる。

【０２０８】さらなるフラグ、辺の交差によってアクティブ化することのできるｓｒｃ＿ｉｓ＿ｌｅａｆ＿ｎｏｄｅ用のフラグと、ｄｅｓｔ＿ｉｓ＿ＰＡＧＥ（常にアクティブ）用のフラグが、有用になる可能性があることに、さらに留意されたい。ｄｅｓｔ＿ｉｓ＿ＰＡＧＥの場合、Ｓ∩Ｄ¯の場合は絶対に発生しないので、これを無視することが可能である。

【０２０９】上では、アクティビティ・テーブル８２０が、エクスプレッションツリー８３０の構造に基づいてどのように構築され、その項目が、ツリー８３０のさまざまなオペランドの変化するアクティブ化を介してどのように完了される（なcouchも書き込まれる）かを示した。テーブル８２０の具体的な例では、異なるアクティブ化と可能な結果を説明するために、７２（＝２×２×３×３×２）個のスタック構造が生じる可能性がある。この形で、条件８０２、８０６、８１２、８１４、８１６および８１８の論理評価が、特定のレベルの適当なスタック演算として識別される正しいアクティビティ８０４をもたらす。

【０２１０】代替実装では、独立のテーブルとして構成されるのではなく、アクティビティ・テーブル８２０を、レベル・アクティブ化テーブル５３０に合併して、組み合わされたテーブル８３０を与えることができる。これによって、データの複製が行われなくなると同時に、優先順位エンコーダ５１２および５１４が正しい辺優先順位だけではなく、アクティブ化演算も選択できるようになり、後者は、モジュール６００から導出された塗潰し色を使用する合成モジュール７００による評価のために画素合成スタック３８に転送（漸進的に）される。このような配置を、図２０Ｇに機能的に図示する。

【０２１１】その代わりに、図２０Ｈに機能的に示されているように、アクティビティ・テーブル８２０が、レベル・アクティブ化テーブル５３０の前にあってもよい。このような場合には、塗潰しカウントとクリップ・カウントの列を、アクティビティ・テーブル８２０に含め、レベル・アクティブ化テーブル５３０から省略することができる。図２０Ｉに機能的に示されているもう１つの代替構成では、アクティビティ・テーブル８２０が、レベル・アクティブ化テーブル５３０の後にある。この場合、塗潰しカウントとクリップ・カウントは、レベル・アクティブ化テーブル５３０に含まれるので、アクティビティ・テーブル８２０から省略することができる。アクティビティ・テーブル８２０が図２０Ａに示されるように構成されるいくつかの応用例では、レベル・アクティブ化テーブル５３０を省略することができる。

【０２１２】アクティビティ・テーブル８２０およびスタック３８に関して上で説明した演算コードは、表示リストから、具体的には命令ストリーム１４（図３参照）から導出される。演算コードは、図３に示された処理ステージのパイプラインを介して、他のデータ（たとえば辺交差、塗潰しデータなど）と共に並列に転送され、画素合成モジュール７００は、優先順位決定、レベル・アクティブ化および塗潰し決定の結果として決定される順序でｏｐコードをスタックに置く。

【０２１３】画素出力モジュール８００の動作を、これから説明する。着信メッセージは、画素出力ＦＩＦＯから読み取られ、これには、画素出力メッセージ、反復メッセージおよびスキャン・ラインの終りメッセージが含まれ、順番に処理される。

【０２１４】画素出力メッセージを受け取った時に、画素出力モジュール８００は、画素を記憶し、その画素をその出力に転送する。反復メッセージを受け取った時には、最後に記憶した画素が、反復メッセージからのカウントによって指定される回数だけ、出力８９８に転送される。スキャン・ラインの終りメッセージを受け取った時には、画素出力モジュール８００は、そのメッセージをその出力に渡す。

【０２１５】出力８９８は、必要に応じて、画素画像データを使用する装置に接続することができる。このような装置には、ビデオ表示ユニットまたはプリンタなどの出力装置、または、ハード・ディスク、ライン・ストア、バンド・ストアまたはフレーム・ストアを含む半導体ＲＡＭなどのメモリ記憶装置、または、コンピュータ・ネットワークが含まれる。

【０２１６】ここまでに説明してきたシステムの実施に伴う困難の１つが、多くの場合に、このシステムが、関連する不透明度値を有する画素を含むオブジェクトの合成を適切に処理しないことである。具体的に言うと、いくつかの状況で、スキャン・ラインに沿った、オブジェクトのうちの１つまたは複数がアクティブでない位置において、重なり合うオブジェクトが不正に合成される。この実施形態でのその状況は、ポーターダフ合成に伴って生じるので、その特定の合成方式に関して説明す

る。しかし、下で詳細に説明する本発明の態様は、他の合成方式の下で生じる同様の困難にも適用可能であることを諒解されたい。

【０２１７】たとえば、式ＰＡＧＥ←（Ａ ｉｎ Ｂ）ｏｖｅｒ（Ｃ ｏｕｔ Ｄ）ｏｖｅｒ ＰＡＧＥは、図２２に示された２進木構造１０００として表すことができる。ここで、Ａ、Ｂ、ＣおよびＤは、辺、表示優先順位および不透明度データによって定義される異なるグラフィカル・オブジェクトである。この例のレベル・アクティブ化テーブル１００１を図２３に示す。図２３では、行１００２で、図２２に示された式のＣ ｏｕｔ Ｄ部分が実施されることがわかる。残念ながら、この構成を使用すると、Ｄがある画素で非アクティブの場合に、ｏｕｔ演算が、Ｄではなくｐａｇｅに対して不正に実行される。Ｃが不透明でない場合には、これが、レンダリング時に視覚的に不正な外見をもたらす。オブジェクトが潜在的に１未満の不透明度を有する場合のこの種の問題を回避するためには、非アクティブ・オブジェクトが、合成式内でｏｖｅｒ演算の左側以外の位置に現れてはならない。ほとんどのオブジェクトが、それらが現れるすべてのスキャン・ラインの少なくとも一部で非アクティブになるとすると、これは大問題になる可能性がある。

【０２１８】この問題を克服する方法の１つが、透明画素を使用してオブジェクトを「バンド・アウト」することである。この形では、各オブジェクトが、効果的に、合成されるオブジェクト群が存在するすべての点に存在するので、ポーターダフ式が正しく適用される。これを上の例に適用すると、２進木１０００の各葉ノード（すなわちＡ、Ｂ、ＣおよびＤ）は、図２４および図２５に示されるように、オブジェクトの対の和集合を囲む境界ボックスのサイズの透明（ガラス）オブジェクトの上に（ｏｖｅｒ）そのオブジェクトを置く式に置換される。

【０２１９】図２４では、２つの透明なガラス境界ボックスＧ１およびＧ２が示されている。Ｇ１は、ＡおよびＢの境界ボックスを表し、Ｇ２は、ＣおよびＤの境界ボックスを表す。エクスプレッションツリー１０００でこれを実施すると、図２５に示された変更されたエクスプレッションツリー１００４がもたらされる。対応するレベル・アクティブ化テーブル１００５を図２６に示す。レベル・アクティブ化テーブル１００５と変更されたエクスプレッションツリー１００４から明白になるとおり、この変更は、各画素に必要なスタック演算の数のかなりの増加をもたらす。これは、スタック演算が、合成される画素の対の一方または両方が透明の場合に実行されるからである。最終的に達成される、実際にレンダリングされる出力に関しては、これらの演算は、スタック資源とプロセッサ時間について比較的浪費的である。

【０２２０】演算の数を減らす方法の１つは、ガラス・オブジェクトが合成される葉ノード・オブジェクトを用いてガラス・オブジェクトをクリップ・アウトすることである。その場合、葉ノードと透明オブジェクトの間のｏｖｅｒ演算は、もはや不要になる。クリッピング動作が、一般に、複数の合成演算よりプロセッサ集中的でないという事実に鑑みて、これは、より効率的なスタックの操作をもたらす。

【０２２１】ＡおよびＢによる透明ボックスＧ１のクリッピングとＣおよびＤによる透明ボックスＧ２のクリッピングを使用する、上の例のレベル・アクティブ化テーブル１００６を、図２７に示す。この実装では、１画素について、複数のｏｖｅｒ演算を使用してＡ、Ｂ、ＣおよびＤがパディングされる場合の１２個のスタック演算と比較して、８つのスタック演算だけが実行されることに留意されたい。これらのレベルのうちの４つは、クリッピング・レベル９９９である。しかし、それでも透明画素を用いる複数の冗長な合成演算が必要であることを諒解されたい。

【０２２２】１つまたは複数の重なり合うオブジェクトが不透明でない場合にポーターダフ合成を正しく実施する好ましい方法を、これから説明する。図２８に示された例を参照すると、演算Ｃ ｏｕｔ Ｄは、領域Ｃ∩Ｄ‾、Ｃ∩ＤおよびＣ‾∩Ｄのそれぞれについて１つずつの、３つの形で表現できる。領域Ｃ∩Ｄ‾は、Ｃによって表現できる。というのは、その領域でｏｕｔ演算に関してＤからの寄与がなく、Ｃが必要だからである。領域Ｃ∩Ｄでは、ＣとＤの両方がアクティブであり、したがって、Ｃ ｏｕｔ Ｄ演算を使用する合成が必要である。領域Ｃ‾∩Ｄでは、ＣまたはＤからの寄与がないので、この領域は、その領域内の画素について合成スタックを構築する時に検討する必要がない。

【０２２３】Ｃ ｏｕｔ Ｄ演算を簡略化した表現を、図４２および図４３に示す。図４２には、ｏｕｔ演算が実行されるオブジェクトＣおよびＤが示されている。上で説明したように、スキャン・ラインは、３つの領域Ｃ∩Ｄ‾、Ｃ∩ＤおよびＣ∩Ｄ‾に分割できる。この場合では、Ｃが不透明でない場合に、領域Ｃ∩Ｄ‾内で、ｏｕｔ演算がＰＡＧＥに対して不正に実行される。

【０２２４】図４３に、前に説明した解決策を示す。オブジェクトのそれぞれが、それがアクティブになる領域にクリッピングされていることに留意されたい。領域Ｃ∩Ｄ‾は、単にレベルＣ１によって表され、領域Ｃ∩Ｄは、レベルＣ１およびＤ１によって表され、領域Ｃ‾∩Ｄは、どちらのレベルからの入力も必要としない。必要な領域だけへのオブジェクトのクリッピングを制御するために特定の演算に固有の必要条件を使用することによって、透明のパディング画素がもはや不要になるので、かなり少ない数の合成演算がもたらされることを諒解されたい。

【０２２５】この例のクリッピングを実施するためのレベル・アクティブ化テーブル１００８を、図２９に示

す。前に説明した方法と比較して、レベル・アクティブ化テーブル１００８では、４つの追加レベルが必要であることがわかる。しかし、これら４つの追加レベルのうちの３つは、クリッピング・レベル１０１５であり、これは、通常は合成演算より集中的でない。また、各画素について作られる実際の合成スタックは、平均して、前に説明した方法の場合より小さい。これは、合成演算が、関連する領域内でのみ行われることを保証するために、クリッピング演算が使用されているという事実に起因する。要するに、プロセッサ時間は、最終的な画像に寄与しない画素の合成に浪費されない。

【０２２６】非自明（non-trivial）ポーターーダフ合成演算子のための合成スタック演算を、図３０および図３１に示す。図３０に示されたテーブル１０１０では、Ｄがアクティブの場合にはＤがすでにスタックの最上部にあり、合成演算子は、それが適用される領域に従って分割されると仮定する。アルファ・チャネル（不透明度）演算は、Ｓの辺上でアクティブ化し、Ｄの辺を用いてクリッピングすることによって、交差領域Ｓ∩Ｄに制限されることに留意されたい。アスタリスクでマークされた事例では、Ｄに寄与するオブジェクトのすべてが、Ｓの辺によってクリップ・インされる必要がある。

【０２２７】図３１のテーブル１０１６は、Ｓ画素値が、それがアクティブになる領域内で、すでにスタック上にある場合の演算を提供する。そのような場合には、Ｄ画素値は、Ｄもアクティブである次の下位レベルにある。アスタリスクでマークされた事例では、スタックは不正な状態になり、その状態が発生しないようにするために「アクティブな枝」のオブジェクトを「非アクティブ」な枝のオブジェクトによってクリッピングするか、最上部の値をスタックからポップするかのいずれかが必要になる。

【０２２８】複雑な式の場合、オブジェクトが複数の他のオブジェクトから構築され、その結果、領域ＳおよびＤの形状を、それらの構築に使用された形状から判定する必要が生じる場合がある。このより複雑な事例を実施するのに必要なステップは、ポーターーダフ合成演算の当業者には明白である。

【０２２９】さまざまな合成演算によって作られる結果のアクティブ領域を、図３２に示されたテーブル１０１１に示す。

【０２３０】好ましい実施形態を使用してレンダリングのためにエクスプレッションツリーを分析する場合には、アプリケーションは、効果的に木の枝ノードのそれぞれのアクティブ領域を決定し、それを木の次の上位ノードに渡す必要がある。図２２に示されたエクスプレッションツリーの詳細な分析を、図３３ないし図４１に関して、この方針に沿って行う。

【０２３１】図３３に、各ノードに関連する演算から生じるアクティブ領域に基づく、図２２に示されたエクスプレッションツリー１０００の予備分析を示す。図３４を参照すると、Ａ ｉｎ Ｂ式が、アクティビティにおいてＡとＢの交差領域に制限されることがわかる。したがって、オブジェクトＡの辺が、オブジェクトＢのクリッピングに使用され、その結果、不要な場合にスタックに色が残されなくなる。同様に、ｉｎ演算を実行するレベルは、オブジェクトＢによる交差領域に制限される。これを達成するのに必要なスタック演算を、図３５に示す。

【０２３２】オブジェクトＢの辺を使用して両方のレベルをアクティブ化し、オブジェクトＡを用いてこの両方のレベルをクリッピングすることによって、クリッピング・レベルを節約しながら、同一の効果を得ることができる。しかし、エクスプレッションツリーのより上位のオブジェクトが、この両方のクリッピング・オブジェクトを必要とする。

【０２３３】次の枝は、図３６に示されるｏｖｅｒ演算であり、この図には、２つの枝のアクティブ領域が示されている。検討しなければならない領域を、図３７に示す。図３７の右側の箱に囲まれたテキスト１０１２は、左側に示される対応する領域１０１３のそれぞれのスタックの状態を示す。

【０２３４】後者の２つの場合、ｏｖｅｒ演算の結果がすでにスタックにある。したがって、交差領域は、さらに演算が必要な領域だけになる。結果のレベル・アクティブ化テーブル項目を、図３８に示す。オブジェクトＣがレベルＯ１をアクティブ化し、AおよびＢが、それを領域Ａ∩Ｂ∩Ｃにクリッピングすることがわかる。この演算全体としては、領域（Ａ∩Ｂ）ＵＣ内の画素に寄与する。

【０２３５】この領域は、エクスプレッションツリーの次のノードに渡されるが、そのノードは、最後のｏｖｅｒであり、これによって、この式がＰＡＧＥに合成される。２つの枝のそれぞれについて検討しなければならない関連領域を図４０に示す。図４０では、やはり、右側の箱に囲まれたテキスト１０１４に、左側に示される対応するアクティブ領域１０１５のスタックの状態を示す。

【０２３６】和集合演算の画素スタックの正しい動作を得ることは、多少の労力を必要とする。１つの解決策は、区域を相互に排他的な区域に分割し、同一の演算のために２つのレベルを使用することである。これは、前のノードの枝のそれぞれのアクティブ領域を使用して、レベルのうちの１つをアクティブ化することによって行うことができる。レベルのうちの１つのアクティブ領域は、他方のレベルのクリップ・アウトに使用される。たとえば、当業者に瞭解されるように、区域Ｃと区域Ａ∩Ｂ∩Ｃ‾を使用して、（Ａ∩Ｂ）ＵＣを作ることができる。

【０２３７】この例の式のレベル・アクティブ化テーブ

ルの結果の項目１０２０を、図４１に示す。レベルＯ２（オブジェクトＣによってアクティブ化される）およびＯ３（オブジェクトＡによってアクティブ化され、Ｂによってクリップ・インされ、Ｃによってクリップ・アウトされる）を組み合わせて、領域（Ａ∩Ｂ）ＵＣに対する最終的なｏｖｅｒ演算をクリッピングする。

【０２３８】クリッピング・レベルは、画素の合成に寄与しないことに留意されたい。寄与するレベルの数、したがって、各画素のスタック演算の数は、平均して、レベルが透明画素によってパッド・アウトされる場合の方法よりかなり少ない。また、所与のスキャン・ライン内で特定のレベルがアクティブになる可能性があると期待される画素の数も減る。

【０２３９】多数のレベルにわたるオブジェクトの合成も、本明細書に記載の方法の外挿によって可能であることを、当業者は認解するであろう。さらに、示されたさまざまな操作を、フレームストア・ベース・システムおよび他のスタック・ベース、ライン・ベースまたはパンド・ベースの合成システムを含む異なる合成パラダイムで使用することができることも認解されるであろう。

【０２４０】前述から、本明細書に記載の方法および装置が、レンダリング処理中の画素画像データの中間記憶を必要とせずに、洗練されたグラフィック記述言語が必要とする機能性のすべてを備えたグラフィック・オブジェクトのレンダリングを提供することは、明白であろう。

【０２４１】前述は、本発明の複数の実施形態を記述したものであり、本発明の趣旨および範囲と、ここまでに付加されたさまざまな態様から逸脱せずに変更を行うことができる。

【図面の簡単な説明】

【図１】好ましい実施形態を組み込まれるコンピュータ・システムの概略ブロック図である。

【図２】好ましい実施形態の機能データ・フローを示すブロック図である。

【図３】好ましい実施形態の画素シーケンシャル・レンダリング装置と、関連する表示リストおよび一時ストアの概略ブロック図である。

【図４】図２の辺処理モジュールの概略機能図である。

【図５】図２の優先順位決定モジュールの概略機能図である。

【図６】図２の塗潰しデータ決定モジュールの概略機能図である。

【図７Ａ】ソースとデスティネーションの間の画素の組合せを示す図である。

【図７Ｂ】ソースとデスティネーションの間の画素の組合せを示す図である。

【図７Ｃ】ソースとデスティネーションの間の画素の組合せを示す図である。

【図８】好ましい実施形態の演算を説明するための例として使用される、オブジェクトが２つある画像を示す図である。

【図９Ａ】図８のオブジェクトのベクトル辺を示す図である。

【図９Ｂ】図８のオブジェクトのベクトル辺を示す図である。

【図１０】図８の画像の複数のスキャン・ラインのレンダリングを示す図である。

【図１１】図８の画像の辺レコードの配置を示す図である。

【図１２Ａ】図１０の例のために図４の配置によって実施される辺更新ルーチンを示す図である。

【図１２Ｂ】図１０の例のために図４の配置によって実施される辺更新ルーチンを示す図である。

【図１２Ｃ】図１０の例のために図４の配置によって実施される辺更新ルーチンを示す図である。

【図１２Ｄ】図１０の例のために図４の配置によって実施される辺更新ルーチンを示す図である。

【図１２Ｅ】図１０の例のために図４の配置によって実施される辺更新ルーチンを示す図である。

【図１２Ｆ】図１０の例のために図４の配置によって実施される辺更新ルーチンを示す図である。

【図１２Ｇ】図１０の例のために図４の配置によって実施される辺更新ルーチンを示す図である。

【図１２Ｈ】図１０の例のために図４の配置によって実施される辺更新ルーチンを示す図である。

【図１２Ｉ】図１０の例のために図４の配置によって実施される辺更新ルーチンを示す図である。

【図１２Ｊ】図１０の例のために図４の配置によって実施される辺更新ルーチンを示す図である。

【図１３Ａ】奇数－偶数塗潰し規則と非ゼロ・ワインディング塗潰し規則を示す図である。

【図１３Ｂ】奇数－偶数塗潰し規則と非ゼロ・ワインディング塗潰し規則を示す図である。

【図１４Ａ】Ｘ座標の大きな変化がスピル条件にどのように寄与し、それらがどのように処理されるかを示す図である。

【図１４Ｂ】Ｘ座標の大きな変化がスピル条件にどのように寄与し、それらがどのように処理されるかを示す図である。

【図１４Ｃ】Ｘ座標の大きな変化がスピル条件にどのように寄与し、それらがどのように処理されるかを示す図である。

【図１４Ｄ】Ｘ座標の大きな変化がスピル条件にどのように寄与し、それらがどのように処理されるかを示す図である。

【図１４Ｅ】Ｘ座標の大きな変化がスピル条件にどのように寄与し、それらがどのように処理されるかを示す図である。

【図１５Ａ】図５の配置によって実施される優先順位更

新ルーチンを示す図である。

【図１５Ｂ】図５の配置によって実施される優先順位更新ルーチンを示す図である。

【図１５Ｃ】図５の配置によって実施される優先順位更新ルーチンを示す図である。

【図１５Ｄ】図５の配置によって実施される優先順位更新ルーチンを示す図である。

【図１５Ｅ】図５の配置によって実施される優先順位更新ルーチンを示す図である。

【図１６Ａ】２つの従来技術の辺記述フォーマットと、好ましい実施形態で使用される辺記述フォーマットを比較するための図である。

【図１６Ｂ】２つの従来技術の辺記述フォーマットと、好ましい実施形態で使用される辺記述フォーマットを比較するための図である。

【図１６Ｃ】２つの従来技術の辺記述フォーマットと、好ましい実施形態で使用される辺記述フォーマットを比較するための図である。

【図１７Ａ】エクスプレッションツリーとして示される単純な合成式と対応する図を示す図である。

【図１７Ｂ】エクスプレッションツリーとして示される単純な合成式と対応する図を示す図である。

【図１８Ａ】正確なスタック演算を保証するために構成されたテーブルを示す図である。

【図１８Ｂ】図１８Ａのテーブルの好ましい形態を示す図である。

【図１９】例のエクスプレッションツリーを示す図である。

【図２０Ａ】図１９の式のアクティビティ・テーブル評価と、そのような評価の間の対応する合成スタックを示す図である。

【図２０Ｂ】図１９の式のアクティビティ・テーブル評価と、そのような評価の間の対応する合成スタックを示す図である。

【図２０Ｃ】図１９の式のアクティビティ・テーブル評価と、そのような評価の間の対応する合成スタックを示す図である。

【図２０Ｄ】図１９の式のアクティビティ・テーブル評価と、そのような評価の間の対応する合成スタックを示す図である。

【図２０Ｅ】図１９の式のアクティビティ・テーブル評価と、そのような評価の間の対応する合成スタックを示す図である。

【図２０Ｆ】図１９の式のアクティビティ・テーブル評価と、そのような評価の間の対応する合成スタックを示す図である。

【図２０Ｇ】アクティビティ・テーブルと関連モジュールのさまざまな構成を示す図である。

【図２０Ｈ】アクティビティ・テーブルと関連モジュールのさまざまな構成を示す図である。

【図２０Ｉ】アクティビティ・テーブルと関連モジュールのさまざまな構成を示す図である。

【図２１】複数の合成演算の結果を示す図である。

【図２２】オブジェクトＡ、Ｂ、ＣおよびＤに対して一連のポーターダフ合成演算を実施するためのエクスプレッションツリーを示す図である。

【図２３】図２２に示された２進木構造を実施するためのレベル・アクティブ化テーブルを示す図である。

【図２４】透明なガラス・オブジェクトの上に置かれた、図２２に示された２進木のオブジェクトを示す図である。

【図２５】図２４に示された配置を実施するための、変更されたエクスプレッションツリーを示す図である。

【図２６】図２５のエクスプレッションツリーを実施するためのレベル・アクティブ化テーブルを示す図である。

【図２７Ａ】透明な箱がＡ、Ｂ、ＣおよびＤによってクリッピングされる、図２４および図２５に示された例の代替のレベル・アクティブ化テーブルを示す図である。

【図２７Ｂ】透明な箱がＡ、Ｂ、ＣおよびＤによってクリッピングされる、図２４および図２５に示された例の代替のレベル・アクティブ化テーブルを示す図である。

【図２８】演算Ｃ　ｏｕｔ　Ｄのエクスプレッションツリーを示す図である。

【図２９Ａ】図２８に示されたクリッピングを実施するためのレベル・アクティブ化テーブルを示す図である。

【図２９Ｂ】図２８に示されたクリッピングを実施するためのレベル・アクティブ化テーブルを示す図である。

【図３０】自明でないポーターダフ合成演算子のための合成スタック演算を示す図である。

【図３１】自明でないポーターダフ合成演算子のための合成スタック演算を示す図である。

【図３２】さまざまな演算子のアクティブ領域を示す図である。

【図３３】式の実施に使用されるステップの予備分析を含む、図２２に示されたものに類似のエクスプレッションツリーを示す図である。

【図３４】図３３に示されたエクスプレッションツリーからのＡ　ｉｎ　Ｂ式のアクティブ領域を示す図である。

【図３５】図３４に示されたｉｎ演算を実行するのに必要なスタック演算を示す図である。

【図３６】下のｉｎ演算およびｏｕｔ演算からのアクティブ領域を示す、図３３のｏｖｅｒ演算の詳細を示す図である。

【図３７】図３６の式に関連して検討される領域のそれぞれのスタックの状態を示す図である。

【図３８】図３６および図３７に示された演算を実施するためのレベル・アクティブ化テーブル項目を示す図である。

【図３９】下のｏｖｅｒ式およびｐａｇｅ式からのアクティブ領域を示す、図３３のエクスプレッションツリーからの最終的なｏｖｅｒ演算を示す図である。

【図４０】図３９の式で検討される領域を示す図である。

【図４１Ａ】図３９に示された式を実施するためのレベル・アクティブ化テーブルの結果の項目を示す図である。

＊【図４１Ｂ】図３９に示された式を実施するためのレベル・アクティブ化テーブルの結果の項目を示す図である。

【図４２】アクティブ領域へのクリッピングの実施の前のＣ　ｏｕｔ　Ｄ演算を示す図である。

【図４３】個々のレベルのアクティブ領域へのクリッピングの後のＣ　ｏｕｔ　Ｄ演算を示す図である。

【図１】

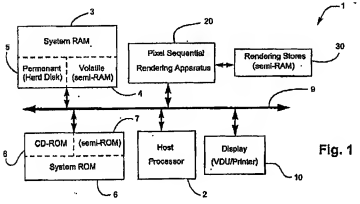


Fig. 1

【図２】

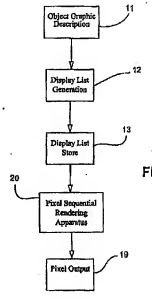


Fig. 2

【図３】

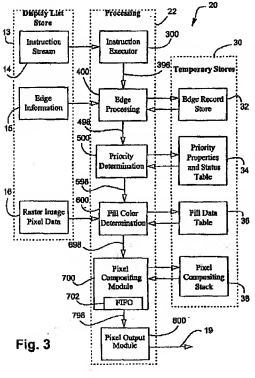


Fig. 3

【図２０Ｂ】

Fig. 20B

【図７Ａ】

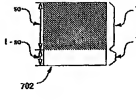


Fig. 7A

【図１０】

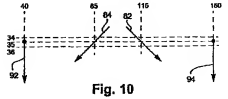


Fig. 10

【図４】

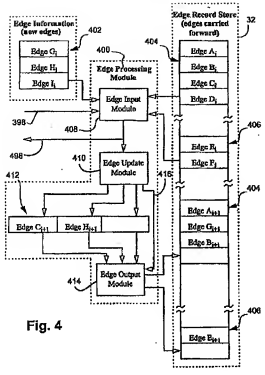


Fig. 4

【図５】

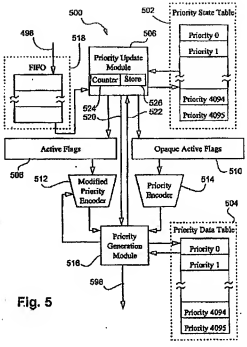


Fig. 5

【図６】

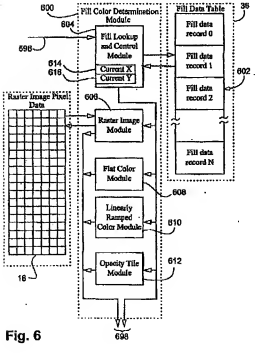


Fig. 6

【図７Ｂ】

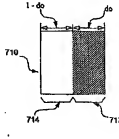


Fig. 7B

【図７Ｃ】

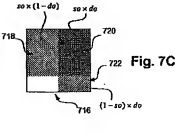


Fig. 7C

【図８】

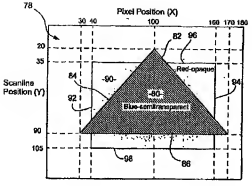


Fig. 8

【図９Ａ】

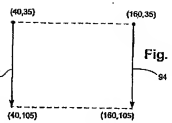


Fig. 9A

【図９Ｂ】

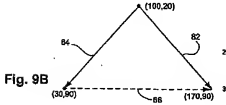


Fig. 9B

【図１１】

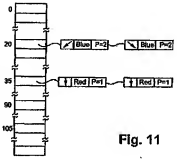


Fig. 11

【図１２Ａ】

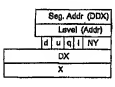


Fig. 12A

【図１３Ａ】

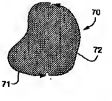


Fig. 13A

【図１２Ｂ】

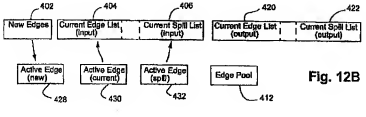


Fig. 12B

【図１２Ｃ】

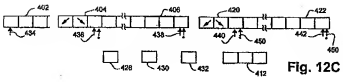


Fig. 12C

【図１２Ｄ】

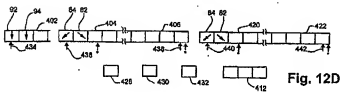


Fig. 12D

【図１２Ｅ】

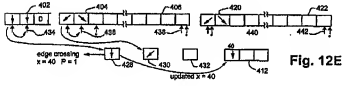


Fig. 12E

【図１２Ｆ】

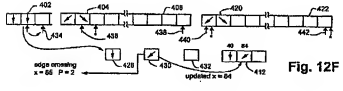


Fig. 12F

【図１２Ｇ】

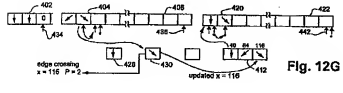


Fig. 12G

【図１６Ａ】

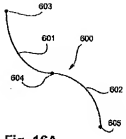


Fig. 16A
(Prior Art)

【図１３Ｂ】

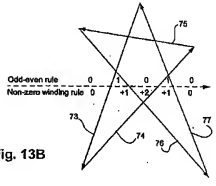


Fig. 13B

【図１２H】

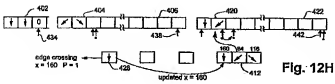


Fig. 12H

【図１２I】

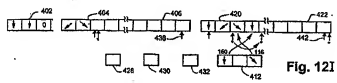


Fig. 12I

【図１２J】

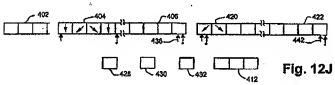


Fig. 12J

【図１４A】

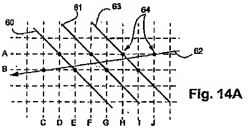


Fig. 14A

【図１４B】

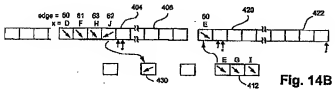


Fig. 14B

【図１６B】

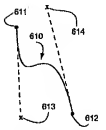


Fig. 16B
(Prior Art)

【図１７A】

Fig. 17A

【図１７B】

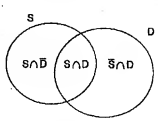


Fig. 17B

【図１４Ｃ】

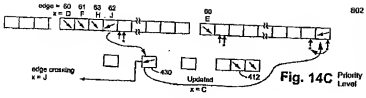


Fig. 14C

【図１８Ａ】

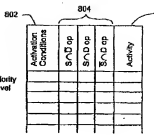


Fig. 18A

【図１４Ｄ】

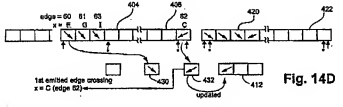


Fig. 14D

【図２０Ｄ】

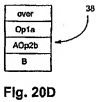


Fig. 20D

【図２０Ｆ】

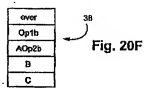


Fig. 20F

【図１４Ｅ】

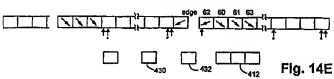


Fig. 14E

【図１５Ａ】

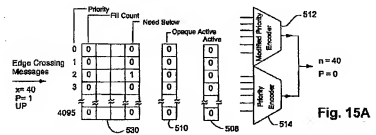


Fig. 15A

【図１５Ｂ】

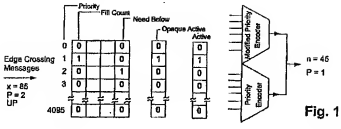


Fig. 15B

【図１５Ｃ】

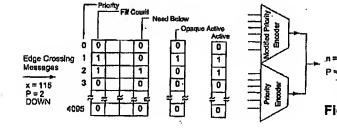


Fig. 15C

【図１５Ｄ】

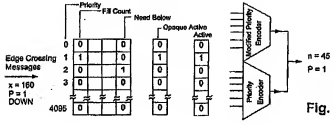


Fig. 15D

【図１５Ｅ】

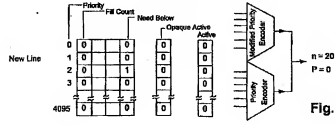
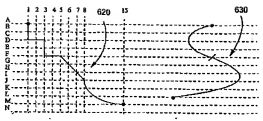


Fig. 15E

【図１６Ｃ】



【図１８Ｂ】

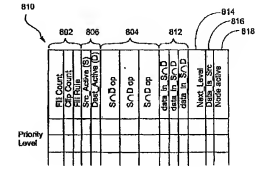
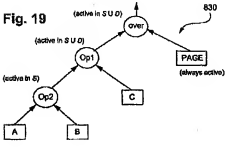


Fig. 18B

【図１９】

Fig. 19



【図２０Ａ】

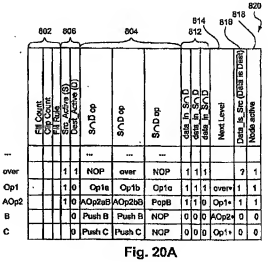


Fig. 20A

【図２０Ｇ】

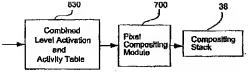


Fig. 20G

【図２０Ｈ】

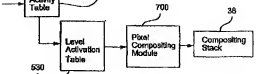


Fig. 20H

【図２０Ｃ】

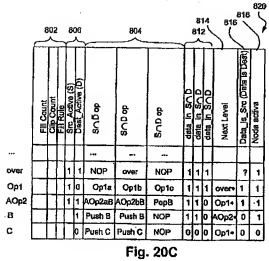


Fig. 20C

【図２０Ｅ】

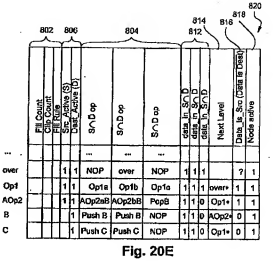


Fig. 20E

【図２０１】

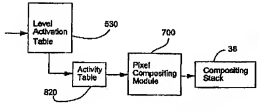


Fig. 20I

【図２１】

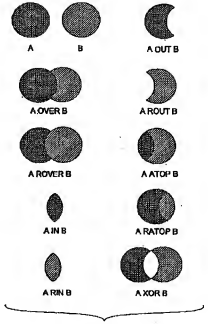


Fig. 21

【図２２】

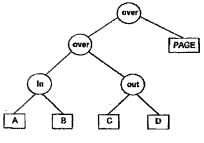


Fig. 22

【図２３】

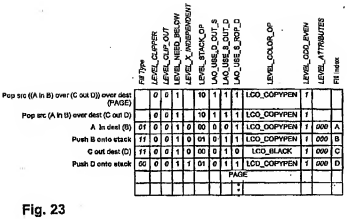


Fig. 23

【図２４】

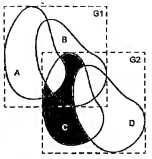


Fig. 24

【図３２】

Operator	Active Region
S over D	S ∪ D
S rover D	S ∪ D
S in D	S ∩ D
S rin D	S ∩ D
S out D	S
S rout D	D
S atop D	D
S ratop D	S
S xor D	S ∪ D

Fig. 32

【図２５】

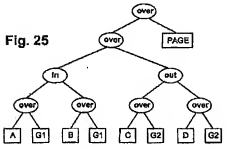


Fig. 25

【図２６】

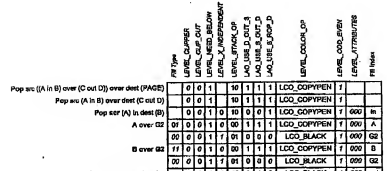


Fig. 26

【図３６】

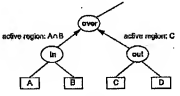
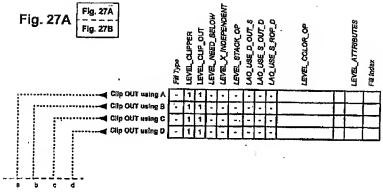


Fig. 36

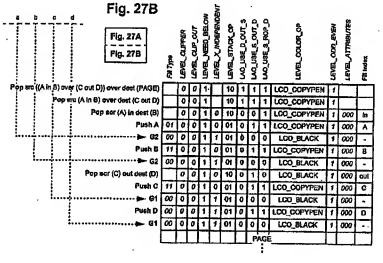
【図２７Ａ】

Fig. 27A



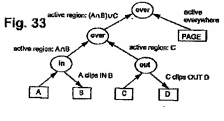
【図２７Ｂ】

Fig. 27B



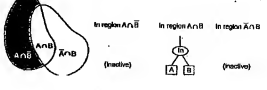
【図３３】

Fig. 33



【図３４】

Fig. 34



【図２８】

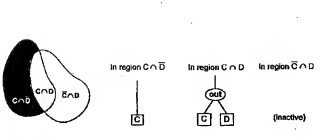


Fig. 28

【図３７】

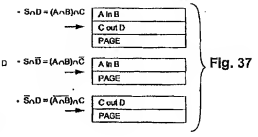


Fig. 37

【図２９Ａ】

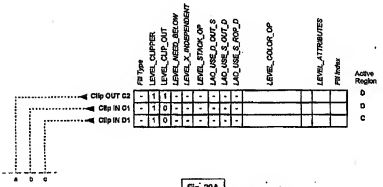


Fig. 29A
Fig. 29B

Fig. 29A

【図３９】

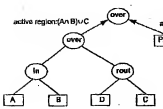


Fig. 39

【図４０】

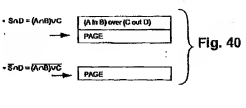


Fig. 40

【図２９Ｂ】

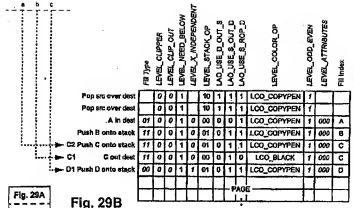


Fig. 29A

Fig. 29B

【図３０】

Operator	Region	Compositing Stack Operations: LEVEL_COLOR_OP	B_INP_D	S_OUT_D	D_OUT_S	FB Object
S over D	S∩D	Result is on the Stack				
	S∩D	LCO_COPYPEN	1	1	1	S
	S∩D̄	LCO_COPYPEN	1	1	0	S
S rover D	S∩D	Result is on the Stack				
	S∩D	LCO_NOP	1	1	1	S
	S∩D̄	LCO_COPYPEN	1	1	0	S
S in D	S∩D	* Clip D with edges of S				
	S∩D	LCO_COPYPEN	1	0	0	S
	S∩D̄	No Operation				
S rin D	S∩D	* Clip D with edges of S				
	S∩D	LCO_NOP	1	0	0	S
	S∩D̄	No Operation				
S out D	S∩D	* Clip D with edges of S				
	S∩D	any	0	1	0	S
	S∩D̄	LCO_COPYPEN	1	1	0	S
S rout D	S∩D	Result is on the Stack				
	S∩D	any	0	0	1	S
	S∩D̄	No Operation				
S atop D	S∩D	Result is on the Stack				
	S∩D	LCO_COPYPEN	1	0	1	S
	S∩D̄	No Operation				
S ratop D	S∩D	* Clip D with edges of S				
	S∩D	LCO_NOP	1	1	0	S
	S∩D̄	LCO_COPYPEN	1	1	0	S
S xor D	S∩D	Result is on the Stack				
	S∩D	any	0	1	1	S
	S∩D̄	LCO_COPYPEN	1	1	0	S

Fig. 30

【図３１】

Operator	Region	Compositing Stack Operations: STACK_OP	LEVEL_COLOR_OP	B_INP_D	S_OUT_D	D_OUT_S
S over D	S∩D	Result on Stack				
	S∩D	10	LCO_COPYPEN	1	1	1
	S∩D̄	Result on stack				
S rover D	S∩D	Result on stack				
	S∩D	10	LCO_NOP	1	1	1
	S∩D̄	Result on Stack				
S in D	S∩D	* Clip D with S edges				
	S∩D	10	LCO_COPYPEN	1	0	0
	S∩D̄	* Clip S with D edges				
S rin D	S∩D	* Clip D with S edges				
	S∩D	10	LCO_NOP	1	0	0
	S∩D̄	* Clip S with D edges				
S out D	S∩D	* Clip D with S edges				
	S∩D	10	any	0	1	0
	S∩D̄	Result on stack				
S rout D	S∩D	Result on stack				
	S∩D	10	any	0	0	1
	S∩D̄	* Clip S with D edges				
S atop D	S∩D	Result on stack				
	S∩D	10	LCO_COPYPEN	1	0	1
	S∩D̄	* Clip D with D edges				
S ratop D	S∩D	* Clip D with S edges				
	S∩D	10	LCO_NOP	1	1	0
	S∩D̄	Result on stack				
S xor D	S∩D	Result on stack				
	S∩D	10	any	0	1	1
	S∩D̄	Result on stack				

Fig. 31

【図３５】

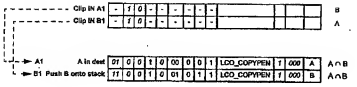


Fig. 35

【図４２】

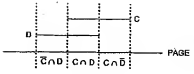


Fig. 42

【図４３】

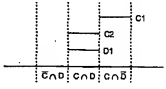


Fig. 43

【図３８】

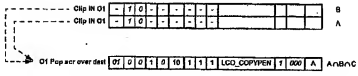


Fig. 38

【図４１Ａ】

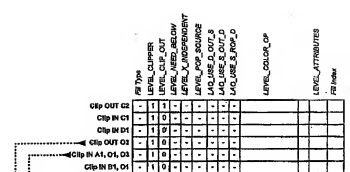


Fig. 41A
Fig. 41B

Fig. 41A

【図４１Ｂ】

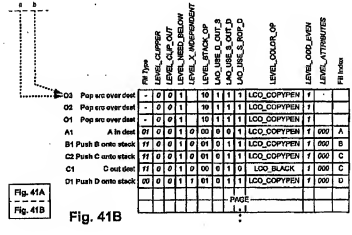


Fig. 41A
Fig. 41B

Fig. 41B

フロントページの続き

(31)優先権主張番号 ＰＰ５８６２
(32)優先日 平成10年９月11日(1998. 9. 11)
(33)優先権主張国 オーストラリア（ＡＵ）
(31)優先権主張番号 ＰＰ９２３４
(32)優先日 平成11年３月16日(1999. 3. 16)
(33)優先権主張国 オーストラリア（ＡＵ）
(31)優先権主張番号 ＰＱ００４９
(32)優先日 平成11年４月29日(1999. 4. 29)
(33)優先権主張国 オーストラリア（ＡＵ）

(72)発明者　クリストファー　フレイザー
オーストラリア国　2113　ニューサウス
ウェールズ州、　ノース　ライド、　トー
マス　ホルト　ドライブ　1、　キヤノン
インフォメーション　システムズ　リサ
ーチ　オーストラリア　プロプライエタリ
ー　リミテッド　内

(72)発明者　ケビン　ムーア
オーストラリア国　2113　ニューサウス
ウェールズ州、　ノース　ライド、　トー
マス　ホルト　ドライブ　1、　キヤノン
インフォメーション　システムズ　リサ
ーチ　オーストラリア　プロプライエタリ
ー　リミテッド　内

【外国語明細書】

1. Title of the Invention

PROCESSING GRAPHIC OBJECTS FOR FAST RASTERISED RENDERING

2. Claims

1. A method of processing graphic objects intended to form a raster pixel image, said processing comprising a process for determining an intersection order between edges of said graphic objects by evaluating corresponding edge records for a current scan line in rasterised display order and determining an edge intersection value for each said edge for a subsequent scan line, said process comprising, during processing of said edge records, the steps of:

retaining a limited number of processed edge records in an unordered first buffer and progressively transferring said processed edge records to a second buffer in order, as orderable processed edge records are added to said first buffer;

transferring unorderable processed edge records to a third buffer in order to order said edge records in said third buffer; and

selectively processing edge records from said second and third buffers for determining ordered intersections for a subsequent scan line.

2. A method according to claim 1, wherein said ordering of said edge records in said third buffer occurs upon adding said edges to said third buffer.

3. A method according to claim 2, wherein said edge records are insertion sorted into said third buffer.

4. A method according to claim 3, wherein edges in said third buffer are ordered on completion of processing of said current scan line.

5. A method according to claim 2, wherein said second and third buffers are combined to form respective portions of a fourth buffer used for said selective processing,

whereby ordered edges from said portions are progressively compared to determine a next edge for said processing.

6. A method according to claim 5, wherein on completion of processing said current scan line, contents of said second and third buffers are swapped to said fourth buffer.

7. A method according to claim 6, wherein said second, third and fourth buffers each include pointers to start and end locations thereof and at an end of said current scan line the pointers are swapped to swap the buffers.

8. A method according to claim 1 wherein said first buffer comprises an edge pool formed in a memory, said method comprising the step of determining from a plurality of active edge records formed in said memory, active edge values for said current scanline and to transfer, for each said active edge value, the corresponding edge record to said edge pool.

9. A method according to claim 8 wherein said second buffer comprises a current edge output list formed in said memory, said method comprising, for each said active edge value, the step of examining records in said edge pool and said corresponding edge record and to transfer an ordered one of said edge records to said current edge list.

10. A method according to claim 9 wherein said third buffer comprises a current spill output list formed in said memory, said method comprising the step of transferring in order to said current spill list, edge records from said edge pool not able to be ordered into said current edge list.

11. A method according to claim 10 wherein said said method comprising, on completion of processing said current scanline, the steps of flushing edge records from said edge pool in order to a corresponding one of said output lists, whereupon said output

lists are assigned as a current edge input list and a current edge input spill list, respectively, for processing of said subsequent scanline, and transferring edge records in order from said input lists to corresponding ones of said active edge records for determination of said active edge values for said subsequent scanline.

12. A method of processing graphic objects intended to form a raster pixel image in a graphic object rendering system, said processing comprising a first process for determining an intersection order between edges of said graphic objects and a current scan line of said raster pixel image, said system comprising:

plural edge records for each of a current scan line and a subsequent scan line, each of said records including a plurality of record locations for retaining at least a pixel location value of a corresponding edge on the corresponding scan line, each of said current and subsequent edge records being divided into at least a main portion and a spill portion, at least the main portion of said current edge records being arranged in raster pixel order;

at least one current active edge record;

a spill active edge record, and

a pool including a limited predetermined number of edge records;

said method comprising the steps of:

(a) transferring a first edge record from each of said main and spill portions of said current edge records into the corresponding active edge records;

(b) comparing values of said active edge records to determine that said active edge record having a lowest value in said raster pixel order and outputting that value and record as a current edge value and record;

(c) updating said current edge record with a value of the corresponding edge for said subsequent scan line;

(d) comparing the updated edge value with edge values within said pool, wherein if the updated edge value is less than an edge value in said pool then

(da) said updated current edge record is transferred to the spill portion of the subsequent edge records; otherwise

(db) (dba) an edge record having a smallest edge value is transferred from said pool to a next record of the main portion of said subsequent edge record; and

(dbb) said updated edge record is transferred to the record of said pool vacated in sub-step (dba); and

(dc) a further edge record is transferred from the corresponding portion of the current edge record to the active edge record vacated by the updated edge record;

(e) repeating steps (b) to (d) until each of said records in said pool are occupied whereupon a smallest edge value record of said pool is transferred to said main portion of said subsequent edge records;

(f) repeating steps (b) to (e) until all records of said current records have been updated, and then flushing records from said pool in order to respective next records within said main portion of said subsequent records;

(g) sorting said records in said spill portion of said subsequent records into raster pixel order;

(h) transferring said subsequent edge records to said current edge records; and

(i) repeating steps (a) to (h) for each further scan line of said raster pixel image.

13. A method according to claim 12, wherein step (g) is performed when step (da) is performed.

14. A method according to claim 12, wherein step (g) is performed by a software sorting routine.

15. A method according to claim 12, wherein step (h) is performed by swapping memory locations pointing to said current edge records and said subsequent edge records.

16. A method according to claim 12, wherein said active edge records further include a new active edge record for receiving from a corresponding new edge records, records of edges commencing on said current scan line, and wherein step (b) comprises comparing each of said new, current and spill active edge records to determine that with said lowest value.

17. A method according to claim 12, wherein step (d), said updated edge value is compared with an immediately preceding updated edge value most recently added to said pool.

18. Apparatus for processing graphic objects intended to form a raster pixel image, said processing comprising a process for determining an intersection order between edges of said graphic objects by evaluating corresponding edge records for a current scanline in rasterised display order and determining an edge intersection value for each said edge for a subsequent scanline, said apparatus comprising:

a memory having an unordered first buffer, a second buffer and a third buffer; and

a processor for retaining a limited number of processed edge records in said unordered first buffer, for progressively transferring said processed edge records to said second buffer in order, as orderable processed edge records are added to said first buffer, for transferring unorderable processed edge records to said third buffer in order to order said edge records in said third buffer, and for selectively processing edge records from said second and third buffers for determining ordered intersections for a subsequent scanline.

19. Apparatus according to claim 18 wherein said first buffer comprises an edge pool, said processor being configured to determine from a plurality of active edge records formed in said memory, active edge values for said current scanline and to transfer, for each said active edge value, the corresponding edge record to said edge pool.

20. Apparatus according to claim 19 wherein said second buffer comprises a current edge output list, said processor being configured, for each said active edge value, to examine records in said edge pool and said corresponding edge record and to transfer an ordered one of said edge records to said current edge list.

21. Apparatus according to claim 20 wherein said third buffer comprises a current spill output list, said processor being configured to transfer in order to said current spill list, edge records from said edge pool not able to be ordered into said current edge list.

22. Apparatus according to claim 21 wherein said processor is configured, on completion of processing said current scanline, for flushing edge records from said edge pool in order to a corresponding one of said output lists, whereupon said output lists are assigned as a current edge input list and a current edge input spill list, respectively, for processing of said subsequent scanline, and for transferring edge records in order from said input lists to corresponding ones of said active edge records for determination of said active edge values for said subsequent scanline.

23. Apparatus for processing graphic objects intended to form a raster pixel image, said processing comprising a process for determining an intersection order between edges of said graphic objects by evaluating corresponding edge records for a current scan line in rasterised display order and determining an edge intersection value for each said edge for a subsequent scan line, said apparatus comprising:

means for retaining a limited number of processed edge records in an unordered first buffer and progressively transferring said processed edge records to a second buffer in order, as orderable processed edge records are added to said first buffer;

means for transferring unorderable processed edge records to a third buffer in order to order said edge records in said third buffer; and

means for selectively processing edge records from said second and third buffers for determining ordered intersections for a subsequent scan line.

24. Apparatus forming part of a graphic object rendering system for processing graphic objects intended to form a raster pixel image, said processing comprising a first process for determining an intersection order between edges of said graphic objects and a current scan line of said raster pixel image, said apparatus comprising:

plural edge records for each of a current scan line and a subsequent scan line, each of said records including a plurality of record locations for retaining at least a pixel location value of a corresponding edge on the corresponding scan line, each of said current and subsequent edge records being divided into at least a main portion and a spill portion, at least the main portion of said current edge records being arranged in raster pixel order;

at least one current active edge record;

a spill active edge record;

a pool including a limited predetermined number of edge records; and an edge record processing arrangement for:

(a) transferring a first edge record from each of said main and spill portions of said current edge records into the corresponding active edge records;

(b) comparing values of said active edge records to determine that said active edge record having a lowest value in said raster pixel order and outputting that value and record as a current edge value and record;

(c) updating said current edge record with a value of the corresponding edge for said subsequent scan line;

(d) comparing the updated edge value with edge values within said pool, wherein if the updated edge value is less than an edge value in said pool then

(da) said updated current edge record is transferred to the spill portion of the subsequent edge records; otherwise

(db) (dba) an edge record having a smallest edge value is transferred from said pool to a next record of the main portion of said subsequent edge record; and

(dbb) said updated edge record is transferred to the record of said pool vacated in sub-step (dba); and

(dc) a further edge record is transferred from the corresponding portion of the current edge record to the active edge record vacated by the updated edge record;

(e) repeating steps (b) to (d) until each of said records in said pool are occupied whereupon a smallest edge value record of said pool is transferred to said main portion of said subsequent edge records;

(f) repeating steps (b) to (e) until all records of said current records have been updated, and then flushing records from said pool in order to respective next records within said main portion of said subsequent records;

(g) sorting said records in said spill portion of said subsequent records into raster pixel order;

(h) transferring said subsequent edge records to said current edge records; and

(i) repeating steps (a) to (h) for each further scan line of said raster pixel image.

25. A computer readable memory medium for storing a program for apparatus which processes graphic objects intended to form a raster pixel image, said processing comprising a process for determining an intersection order between edges of said graphic objects by evaluating corresponding edge records for a current scan line in rasterised display order and determining an edge intersection value for each said edge for a subsequent scan line, said program comprising:

code for a retaining step for retaining a limited number of processed edge records in an unordered first buffer and progressively transferring said processed edge records to a second buffer in order, as orderable processed edge records are added to said first buffer;

code for a transfer step for transferring unorderable processed edge records to a third buffer in order to order said edge records in said third buffer; and

code for a process step for selectively processing edge records from said second and third buffers for determining ordered intersections for a subsequent scan line.

26. A computer readable medium comprising a computer program product for processing graphic objects intended to form a raster pixel image in a graphic object rendering system, said processing comprising a first process for determining an intersection order between edges of said graphic objects and a current scan line of said raster pixel image, said computer program product being associated with:

plural edge records for each of a current scan line and a subsequent scan line, each of said records including a plurality of record locations for retaining at least a pixel location value of a corresponding edge on the corresponding scan line, each of said current and subsequent edge records being divided into at least a main portion and a spill portion, at least the main portion of said current edge records being arranged in raster pixel order;

at least one current active edge record;

a spill active edge record, and

a pool including a limited predetermined number of edge records;

said first process being configured to implement the steps of:

(a) transferring a first edge record from each of said main and spill portions of said current edge records into the corresponding active edge records;

(b) comparing values of said active edge records to determine that said active edge record having a lowest value in said raster pixel order and outputting that value and record as a current edge value and record;

(c) updating said current edge record with a value of the corresponding edge for said subsequent scan line;

(d) comparing the updated edge value with edge values within said pool, wherein if the updated edge value is less than an edge value in said pool then

(da) said updated current edge record is transferred to the spill portion of the subsequent edge records; otherwise

(db) (dba) an edge record having a smallest edge value is transferred from said pool to a next record of the main portion of said subsequent edge record; and

(dbb) said updated edge record is transferred to the record of said pool vacated in sub-step (dba); and

(dc) a further edge record is transferred from the corresponding portion of the current edge record to the active edge record vacated by the updated edge record;

(e) repeating steps (b) to (d) until each of said records in said pool are occupied whereupon a smallest edge value record of said pool is transferred to said main portion of said subsequent edge records;

(f) repeating steps (b) to (e) until all records of said current records have been updated, and then flushing records from said pool in order to respective next records within said main portion of said subsequent records;

(g) sorting said records in said spill portion of said subsequent records into raster pixel order;

(h) transferring said subsequent edge records to said current edge records; and

(i) repeating steps (a) to (h) for each further scan line of said raster pixel image.

27. A method according to claim 1 or 12, said processing comprising a (second) process for reproducing pixel image data defined by at least one said graphic object forming part of said image, said image being rendered in a rasterised fashion, said method comprising the steps of:

(j) establishing a plurality of graphic object fill types distinguishable by at least one property of each type;

(k) receiving a graphic object requiring rendering, determining the corresponding fill type and using the determined fill type to access a filling record associated with said graphic object; and

(l) performing a filling process associated with said filling record to determine said pixel image data in a rasterised fashion corresponding to said graphic object.

28. A method according to claim 27, wherein one of said types is a raster image type and if said object has a fill type identifying a raster image type object, step (i) comprises

using a pixel location in said image to determine a mapping to said pixel image data whereupon that data so mapped is selected for output at said pixel location.

29. A method according to claim 27, wherein said mapping includes an affine transform of said pixel image data.

30. A method according to claim 27 wherein said mapping performs a texture mapping of pixel image data to said raster image.

31. A method according to claim 27, wherein a pixel location in said raster image to be rendered is used to access an entry in a table, said table including a reference to a component of said pixel image data to be rendered within said graphic object, said second process further comprising manipulating said reference to access part of said pixel image data which is then used to contribute to a rendered pixel value of said raster image at said pixel location.

32. A method according to claim 31, wherein the contribution of said pixel image data comprises a single pixel data value that is reproduced in said raster image.

33. A method according to claim 31, wherein the contribution of said pixel image data comprises at least one pixel data value that is composited with at least one value derived from at least one other graphic object within said raster image to derive a single pixel value that is reproduced in said rasterised image.

34. A method according to claim 31, wherein the contribution of said pixel image data comprises one pixel data value that is composited with at least one value derived from at least one other graphic object within said raster image to derive a plurality of pixel values that are reproduced in said rasterised image.

35. A method according to claim 28, wherein said mapping includes interpolating a plurality of pixel image data located about a mapped location within said pixel image data.

36. A method according to claim 27, wherein step (k) comprises the sub-step of:

(ka) developing a stack of values corresponding to each said (different) pixel on a scan line of said raster image from data sourced from a table of compositing properties ordered according to object priorities, said stack having a depth corresponding to a number of said objects active at said pixel location.;

and wherein step (i) comprises the sub-step of:

(ia) determining a pixel value at said pixel location using the corresponding said stack.

37. A method according to claim 1 or 12, wherein each said graphic object is described by a data structure comprising at least one edge that defines at least part of a boundary of said object, said edge including a plurality of segments described by segment descriptions and extending between starting and ending scan lines of said edge characterised in that at least two of said segments are defined by differing data formats.

38. A method according to claim 37, wherein, at a connection between said segments, a value returned by a first rendered one of said segments is used as an initial determining value for the following one of said segments.

39. A method according to claim 37, wherein each said data format includes identifiers for a start and an end of said segment and further parameters permitting determination of said segment.

40. A method according to claim 39, wherein said segment is an orthogonal step segment and said further parameters include a signed x-step value and an unsigned y-step value.

41. A method according to claim 39, wherein said segment is a straight line inclined to a scan line direction and said further parameters include a first order value for adding to a pixel location value of a current scan line to derive a pixel location value for a following scan line.

42. A method according to claim 39, wherein said segment is a quadratic curve and said further parameters include a first order value associated with a current scan line for adding to a pixel location value of said current scan line to derive a pixel location value for a following scan line, and a second order value for adding to the first order value of said current scan line to derive a first order value for the following scan line.

43. A method according to claim 39, wherein said segment is an N-th order polynomial curve and said further parameters include 1 to N order values associated with a current scan line for determining, by adding, a pixel location value of said current scan line to derive a pixel location value for a following scan line, and 1 to (N-1) order values for the following scan line.

44. A method according to claim 1 or 12, said processing comprising a third process for rendering a segment edge, said method including the steps of interpreting plural segment data types and evaluating pixel locations on said edge for each of said segments by manipulating the corresponding segment data types characterised by a seamless transition from one said segment to an adjoining said segment resulting from a terminating pixel location of a first segment is used to determine a following pixel location for the following said segment.

45. A method according to claim 1 or 12, said processing comprising a fourth process for compositing first and second pixel values each having corresponding color and opacity values, said fourth process comprising the steps of:

(j) dividing each of said pixel values to define at least two regions thereof, a first being a fully opaque region and another being a fully transparent region;

(k) determining areas of intersection between each of said regions to thus derive a opacity value for each of the intersecting regions;

(l) selecting at least one of the regions as contributing color and opacity to a composited pixel value;

(m) determining the colors of the selected regions according to a predetermined raster operation; and

(n) forming a first sum of color-opacity products for the selected regions and dividing said first sum by a second sum of the opacities of the selected regions to derive a color value of said composited pixel, said second sum being an opacity of said composited pixel.

46. A method according to claim 45, wherein the regions of one of said first or second pixel values are arranged orthogonal to those of the other said second or first pixel values to define three intersecting regions of interest upon which steps (l) to (n) are applied.

47. A method according to claim 46, wherein said compositing operation comprises identifying at least one of said regions of interest and using said identified region(s) of interest in said sums.

48. A method according to claim 45 wherein any one of steps (j) to (l) comprises the step of normalising said color and opacity values.

49. A method according to claim 48, wherein step (m) comprises using the normalised opacity value to delineate the corresponding said regions.

50. A method according to claim 1 or 12, said processing comprising a fourth process for compositing a source color and opacity (*so*) with a destination color and opacity (*do*), said fourth process comprising the steps of:

(j) normalising the color and opacity values for each of said source and destination to define at least two regions thereof, a first being a fully opaque region {(*so*),(*do*)} and another being a fully transparent region {(1-*so*),(1-*do*)};

(k) orthogonally arranging the destination regions to intersect the source regions to derive a opacity value for each of each of three intersecting regions, those being

(i) source outside destination {*so* x (1-*do*)};

(ii) source intersect destination {*so* x *do*}; and

(iii) destination outside source {(1-*so*) x *do*};

(l) determining a source intersect destination (*sc* x *dc*) color value as an opaque component of a composited pixel value according to a predetermined raster operation; and

(m) determining an opacity component of said composited pixel value using a sum of selected color-opacity products, those being:

sc(*so* x (1-*do*)) , (*so* x *da*)(*sc* rop *dc*) , and *dc*((1-*so*) x *do*),

where rop represents said predetermined raster operation.

51. A method according to claim 50, wherein said color-opacity products are selectable according to a predetermined opacity compositing operation.

52. A method according to claim 51, wherein said predetermined opacity compositing operation comprises a flag corresponding to each of said intersecting regions and said sum is determined as the sum of the areas of those said intersecting regions for which the corresponding said flag is set.

53. A method according to claim 1 or 12, said processing comprising a fourth process for compositing first and second pixel values, each having corresponding color and

opacity values, according to a predetermined compositing operation, to form a composited pixel value, said fourth process comprising the steps of:

(j) dividing each of said pixel values to define at least two regions thereof, a first being a fully opaque region and another being a fully transparent region;

(k) determining areas of intersection between each of said regions to thus derive a opacity value for each of the intersecting regions;

(l) selecting according to said compositing operation at least one of the regions as contributing color and opacity to said composited pixel value;

(m) determining a color of each said selected region according to a predetermined raster operation being part of said compositing operation; and

(n) forming, according to a predetermined opacity operation being part of said compositing operation, a first sum of color-opacity products of said selected regions and dividing said first sum by a second sum of the opacities of said selected regions to derive a color value of said composited pixel value, said second sum being an opacity of said composited pixel, wherein said opacity operation comprises a flag corresponding to each of said regions and said first sum is determined as the sum of the areas of those said regions for which the corresponding said flag is set.

54. A method according to claim 1 or 12, further comprising compositing graphic objects characterised by raster operations between said objects that account for transparency components of each of said objects.

55. A method according to claim 1 or 12, said processing comprising a fifth process for applying a compositing expression to said graphical object environment, said compositing expression having a plurality of priority levels, said fifth process comprising, for each priority level, in ascending priority order, the steps of:

identifying a plurality of alternative actions available to arise from an operation associated with at least one operand of said priority level;

associating said alternate actions with a plurality of activation conditions for said priority level, said activation conditions including an activity state of each said operand; and

logically combining said activation conditions to select one of said alternate actions for determination and application to a next higher priority level.

56. A method according to claim 55, wherein said activation conditions include at least one flag indicating a corresponding condition for which said operation provides data, a pointer to an entry for a subsequent operation which uses said data, and at least one flag to indicate which said operand provides said data.

57. A method according to claim 55, wherein said compositing expression is evaluated by rendering said selected alternate actions in ascending priority order.

58. A method according to claim 57, wherein each said alternate action comprises a stack operation, said stack operations together determining pixel output values for a span of pixels between two object edges on a scan line.

59. A method according to claim 55, wherein said compositing expression is a hierarchically structured representation of said image.

60. A method according to claim 58, wherein said compositing expression is optimised in regard to the number of pixel operations required to render said image represented by said compositing expression.

61. A method according to claim 60, wherein said logical combining of said activation conditions comprises modifying a manner in which said compositing expression is evaluated without modifying said hierarchically structured representation.

62. A method according to claim 59, wherein said image comprises at least a pixel based image component.

63. A method according to claim 55, wherein a wholly opaque graphical object at a particular priority level acts to eliminate one or more objects at lower priority levels of said compositing expression.

64. A method according to claim 54, wherein at least one of said steps comprises forming a table of said alternate actions and said activation conditions.

65. A method according to claim 1 or 12, said processing comprising a fifth process for forming an optimised compositing expression, said compositing expression being a hierarchically structured representation of an arrangement of said graphical objects, wherein each said graphical object has a predetermined outline, said fifth process comprising the steps of:

determining an expression for a plurality of regions, each said region being defined by at least one region outline substantially following at least one of said predetermined outlines or parts thereof;

identifying a plurality of further regions depending on at least one characteristic of at least one of said regions, wherein each said further region has an associated compositing operation;

logically combining said further regions and said associated compositing operations; and

applying said logical combination to a higher level of said hierarchically structured representation to form said optimised compositing expression.

66. A method according to claim 65, wherein at least one of said steps comprises forming a table of said alternate actions and said activation conditions.

67. A method according to claim 1 or 12, said processing comprising a fifth process for forming a stack of compositing operations used for the rendering of said image, said fifth process comprising the steps of:

establishing a table arranged according to a priority level at which each said graphical object is reproducible, said table including for each said priority level:

(i) entries for a plurality of alternate actions available to arise from an operation associated with at least one operand at said priority level; and

(ii) entries for activation conditions including an activity state for each said operand;

updating said entries for a span of displayable pixels between two adjacent object edges on a displayable scan lines and for each said span

analysing said activation conditions to select one of said alternate actions for each said priority level for transfer in ascending priority order to said stack.

68. A method according to claim 1 or 12, said processing comprising a sixth process for generating a compositing stack for use in rendering said image formed of a plurality of graphic objects, each of the graphic objects being described by components including at least an edge and a viewing priority relative to the other graphic objects, the sixth process comprising the steps of:

(j) determining the boundaries of said objects and intersections therebetween;

(k) for each said intersection so determined, providing a level activation table of objects active at said intersection arranged according to viewing priority;

(l) ascertaining whether an active overlying graphic object overlaps an underlying graphic object within an overlap region;

(m) where such an overlap occurs, performing clipping operations using the boundaries of the overlying graphic object and the underlying graphic object to generate clipped versions of the overlying and/or underlying graphic objects within the overlap region and/or clipped versions of the underlying and/or overlying graphic objects outside the overlap region;

(n) adding to the level activation table:

(na) one or more first levels corresponding to those regions of the underlying and overlying graphic objects disposed within the overlap region; and/or

(nb) one or more second levels corresponding to those regions of the underlying and/or overlying graphic objects disposed outside the overlap region; and

(o) for each pixel within said image, determining a compositing stack based on the level activation table entries relevant to that pixel location.

69. A method according to claim 68, wherein step (j) comprises a sub-step (ja) of determining, in a predetermined order, coordinates of intersection of those edges of said objects that intersect said current scan line.

70. A method according to claim 68, wherein:

step (l) comprises a substep (la) of determining a compositing operator which applies to the overlapping graphic objects; and

step (m) comprises a substep (ma) of performing the clipping operations using the edges of the overlying object and/or the underlying object on the basis of the compositing operator determined in substep (la).

71. A method according to claim 70, wherein substep (nb) comprises a substep (nc) of adding to the level activation table the one or more second levels corresponding to the underlying and/or overlying objects, the one or more second levels being selected for addition to the level activation table on the basis of the compositing operator determined in substep (la).

72. A method according to claim 69, wherein said compositing operator is a Porter and Duff compositing operator.

73. A method according to claim 1 or 12, said processing comprising a sixth process for generating a compositing stack based on table entries in a level activation table, for use in rendering said plurality of graphic objects onto said raster pixel image having a plurality of scan lines and a plurality of locations on each scan line, each of the graphic objects being described by components comprising at least an edge and a viewing priority relative to the other graphic objects, the sixth process including the steps of:

(j) determining, in a predetermined order, coordinates of intersection of those edges of said objects that intersect said scan line;

(k) for each edge intersection so determined, updating the level activation table, the level activation table being arranged according to viewing priority;

(l) ascertaining whether an active overlying graphic object overlaps an underlying graphic object within an overlap region;

(m) where such an overlap occurs, performing clipping operations using edges of the overlying graphic object and the underlying graphic object to generate clipped versions of the overlying and/or underlying graphic objects within the overlap region and/or clipped versions of the underlying and/or overlying graphic objects outside the overlap region;

(n) adding to the level activation table:

(na) one or more first levels corresponding to those regions of the underlying and overlying graphic objects disposed within the overlap region; and/or

(nb) one or more second levels corresponding to those regions of the underlying and/or overlying graphic objects disposed outside the overlap region; and

(o) for each pixel location on the scan line, determining a compositing stack based on the level activation table entries relevant to that pixel location.

74. A method according to claim 73, wherein step (m) comprises the sub-step of performing clipping operations using edges of the overlying graphic object and the underlying graphic object to generate clipped versions of the overlying and underlying

graphic objects within the overlap region and a clipped version of the underlying or overlying graphic objects outside the overlap region.

75. A method according to claim 68, wherein said overlying object is non-opaque.

3. Detailed Explanation of the Invention

[Field of Invention]

The present invention relates to the rendering of object graphic elements into raster pixel images and, in particular, to efficient rendering of such elements to pixel image data without the use of frame or line storage of the pixel data as part of the rendering process.

[Background of Invention]

Most object based graphics systems utilise a frame store or page buffer to hold a pixel-based image of the page or screen. Typically, the outlines of the graphic objects are calculated, filled and written into the frame store. For two-dimensional graphics, objects which appear in front of other objects are simply written into the frame store after the background objects, thereby replacing the background on a pixel-by-pixel basis. This is commonly known in the art as the "Painter's algorithm". Objects are considered in priority order, from the rearmost object to the foremost object, and, typically, each object is rasterised in scan line order and pixels are written to the frame store in sequential runs along each scan line.

There are essentially two problems with this technique. The first is that it requires fast random access to all the pixels in the frame store. This is because each new object considered could affect any pixel in the frame-store. For this reason, the frame store is normally kept in semiconductor random access memory (RAM). For high resolution colour printers the amount of RAM required is very large, typically in excess of 100 MBytes, which is costly and difficult to operate at high speed. The second problem

is that many pixels which are painted (rendered), are over-painted (re-rendered) by later objects. Painting the pixels with the earlier objects was a waste of time.

One method for overcoming the large frame-store problem is the use of "banding". When banding is used, only part of the frame-store exists in memory at any one time. All of the objects to be drawn are retained in a "display list". The whole image is rendered as above, but pixel painting (rendering) operations that try to paint (render) outside the fraction of the frame-store which exists are "clipped" out. After all the objects have been drawn, the fractional part of the frame-store is sent to the printer (or some other location) and another fraction of the frame-store is selected and the process repeated. There are penalties with this technique. For example, the objects being drawn must be considered and re-considered many times - once for each band. As the number of bands increases, so too does the repetitious examination of objects requiring rendering. The technique of banding does not solve the problem of the cost of over-painting.

Some other graphic systems consider the image in scan line order. Again, all the objects to be drawn are retained in a display list. On each scan line the objects which intersect that scan line are then considered in priority order and for each object, spans of pixels between object edge intersection points are set in a line store. This technique also overcomes the large frame store problem, but still suffers from the over-paint problem.

There are other techniques which overcome both the large frame-store problem and the over-painting problem. In one such technique, each scan line is produced in turn. Again, all the objects to be drawn are retained in a display list. On each scan line, the edges of objects which intersect that scan line are held in order of increasing coordinate of intersection with the scan line. These points of intersection, or edge crossings, are considered in turn and used to toggle an array of active flags. There is one active flag for each object priority which is of interest on the scan line. Between each pair of edges considered, the colour data for each pixel which lies between the first edge and the next edge is generated by using a priority encoder on the active flags to determine which priority is topmost, and using the colour associated with that priority for the pixels of the span between the two edges. In preparation for the next scan line, the coordinate of

intersection of each edge is updated in accordance with the nature of each edge. Adjacent edges which become mis-sorted as a result of this update are swapped. New edges are also merged into the list of edges.

This technique has the significant advantages that there is no frame store or line store, there is no over painting, and the object priorities are dealt with in constant order time, rather than order N time (where N is the number of priorities).

However, there are several limitations to this technique:

(i) The technique only supports the "odd-even" fill rule, known in the art for determining the inside versus outside state of an object from its edges. The "non-zero winding" fill rule, which is a required feature of many graphic description languages, is not supported by that technique.

(ii) Large mis-sorting can occur for which a simple swapping technique is inadequate to repair. While a brute-force sort of the whole of the edge list on each scan line can be performed, this is very slow.

(iii) The technique does not support raster (pixel-based) images as an object type. Such images are a required feature of most graphic description languages.

(iv) The technique only supports objects which are opaque, where each painted pixel strictly obscures the pixels of objects with lesser priority. The technique does not support raster operations in which the colour of two or more graphic objects interact. Such operations include drawing in XOR mode, or the compositing of partially transpacent objects. These modification operations are a required feature of most graphic description languages.

(v) The technique does not support clipping, where one or more clip shapes suppress some number of other graphics objects inside (or outside) the bounds of the clip shapes. Clipping is a required feature of most graphic description languages.

(vi) The technique uses voluminous and inefficient encoding of object edges, particularly for text. It is desirable for such an extremely common element of graphic descriptions to be represented in a simpler fashion.

(vii) The technique in some instances does not provide for accurate evaluation of complicated compositing expressions where the activity of one or more objections is variable.

The inability of the technique to implement many features required by existing graphic description languages severely limits its use.

Further, some existing rendering interfaces require implementation of the bit-wise logical combination of the colour of two or more graphic objects. Other existing rendering interfaces require implementation of an alpha channel (also referred to as transparency, opacity, or matte) based combination of the colour of two or more graphic objects. Current techniques do not allow these two features to be implemented in a unified fashion.

[Summary of the Invention]

It is an object of the present invention to substantially overcome, or at least ameliorate, one or more deficiencies with prior art systems.

In accordance with one aspect of the present invention there is disclosed a method of processing graphic objects intended to form a raster pixel image, said processing comprising a process for determining an intersection order between edges of said graphic objects by evaluating corresponding edge records for a current scan line in rasterised display order and determining an edge intersection value for each said edge for a subsequent scan line, said process comprising, during processing of said edge records, the steps of:

retaining a limited number of processed edge records in an unordered first buffer and progressively transferring said processed edge records to a second buffer in order, as orderable processed edge records are added to said first buffer;

transferring unorderable processed edge records to a third buffer in order to order said edge records in said third buffer; and

selectively processing edge records from said second and third buffers for determining ordered intersections for a subsequent scan line.

In accordance with another aspect of the present invention there is disclosed a method of processing graphic objects intended to form a raster pixel image in a graphic

object rendering system, said processing comprising a first process for determining an intersection order between edges of said graphic objects and a current scan line of said raster pixel image, said system comprising:

plural edge records for each of a current scan line and a subsequent scan line, each of said records including a plurality of record locations for retaining at least a pixel location value of a corresponding edge on the corresponding scan line, each of said current and subsequent edge records being divided into at least a main portion and a spill portion, at least the main portion of said current edge records being arranged in raster pixel order;

at least one current active edge record;

a spill active edge record, and

a pool including a limited predetermined number of edge records;

said method comprising the steps of:

(a) transferring a first edge record from each of said main and spill portions of said current edge records into the corresponding active edge records;

(b) comparing values of said active edge records to determine that said active edge record having a lowest value in said raster pixel order and outputting that value and record as a current edge value and record;

(c) updating said current edge record with a value of the corresponding edge for said subsequent scan line;

(d) comparing the updated edge value with edge values within said pool, wherein if the updated edge value is less than an edge value in said pool then

(da) said updated current edge record is transferred to the spill portion of the subsequent edge records; otherwise

(db) (dba) an edge record having a smallest edge value is transferred from said pool to a next record of the main portion of said subsequent edge record; and

(dbb) said updated edge record is transferred to the record of said pool vacated in sub-step (dba); and

(dc) a further edge record is transferred from the corresponding portion of the current edge record to the active edge record vacated by the updated edge record;

(e) repeating steps (b) to (d) until each of said records in said pool are occupied whereupon a smallest edge value record of said pool is transferred to said main portion of said subsequent edge records;

(f) repeating steps (b) to (e) until all records of said current records have been updated, and then flushing records from said pool in order to respective next records within said main portion of said subsequent records;

(g) sorting said records in said spill portion of said subsequent records into raster pixel order;

(h) transferring said subsequent edge records to said current edge records; and

(i) repeating steps (a) to (h) for each further scan line of said raster pixel image.

In accordance with another aspect of the present invention there is disclosed apparatus for processing graphic objects intended to form a raster pixel image, said processing comprising a process for determining an intersection order between edges of said graphic objects by evaluating corresponding edge records for a current scanline in rasterised display order and determining an edge intersection value for each said edge for a subsequent scanline, said apparatus comprising:

a memory having an unordered first buffer, a second buffer and a third buffer; and

a processor for retaining a limited number of processed edge records in said unordered first buffer, for progressively transferring said processed edge records to said second buffer in order, as orderable processed edge records are added to said first buffer, for transferring unorderable processed edge records to said third buffer in order to order said edge records in said third buffer, and for selectively processing edge records from said second and third buffers for determining ordered intersections for a subsequent scanline.

In accordance with another aspect of the present invention there is disclosed apparatus for processing graphic objects intended to form a raster pixel image, said

processing comprising a process for determining an intersection order between edges of said graphic objects by evaluating corresponding edge records for a current scan line in rasterised display order and determining an edge intersection value for each said edge for a subsequent scan line, said apparatus comprising:

means for retaining a limited number of processed edge records in an unordered first buffer and progressively transferring said processed edge records to a second buffer in order, as orderable processed edge records are added to said first buffer;

means for transferring unorderable processed edge records to a third buffer in order to order said edge records in said third buffer; and

means for selectively processing edge records from said second and third buffers for determining ordered intersections for a subsequent scan line.

In accordance with another aspect of the present invention there is disclosed a computer readable memory medium for storing a program for apparatus which processes graphic objects intended to form a raster pixel image, said processing comprising a process for determining an intersection order between edges of said graphic objects by evaluating corresponding edge records for a current scan line in rasterised display order and determining an edge intersection value for each said edge for a subsequent scan line, said program comprising:

code for a retaining step for retaining a limited number of processed edge records in an unordered first buffer and progressively transferring said processed edge records to a second buffer in order, as orderable processed edge records are added to said first buffer;

code for a transfer step for transferring unorderable processed edge records to a third buffer in order to order said edge records in said third buffer; and

code for a process step for selectively processing edge records from said second and third buffers for determining ordered intersections for a subsequent scan line.

Other aspects of the present invention will be apparent from the following description.

[Embodiments]

Fig. 1 illustrates schematically a computer system 1 configured for rendering and presentation of computer graphic object images. The system includes a host processor 2 associated with system random access memory (RAM) 3, which may include a non-volatile hard disk drive or similar device 5 and volatile, semiconductor RAM 4. The system 1 also includes a system read-only memory (ROM) 6 typically founded upon semiconductor ROM 7 and which in many cases may be supplemented by compact disk devices (CD ROM) 8. The system 1 may also incorporate some means 10 for displaying images, such as a video display unit (VDU) or a printer, both of which operate in raster fashion.

The above-described components of the system 1 are interconnected via a bus system 9 and are operable in a normal operating mode of computer systems well known in the art, such as IBM PC/AT type personal computers and arrangements evolved therefrom, Sun Sparcstations and the like.

Also seen in Fig. 1, a pixel sequential rendering apparatus 20 connects to the bus 9, and in the preferred embodiment is configured for the sequential rendering of pixel-based images derived from graphic object-based descriptions supplied with instructions and data from the system 1 via the bus 9. The apparatus 20 may utilise the system RAM 3 for the rendering of object descriptions although preferably the rendering apparatus 20 may have associated therewith a dedicated rendering store arrangement 30, typically formed of semiconductor RAM.

Referring now to Fig. 2, a functional data flow diagram of the preferred embodiment is shown. The functional flow diagram of Fig. 2 commences with an object graphic description 11 which is used to describe those parameters of graphic objects in a

fashion appropriate to be generated by the host processor 2 and/or, where appropriate, stored within the system RAM 3 or derived from the system ROM 6, and which may be interpreted by the pixel sequential rendering apparatus 20 to render therefrom pixel-based images. For example, the object graphic description 11 may incorporate objects with edges in a number of formats including straight edges (simple vectors) that traverse from one point on the display to another, or an orthogonal edge format where a two-dimensional object is defined by a plurality of edges including orthogonal lines. Further formats, where objects are defined by continuous curves, are also appropriate and these can include quadratic polynomial fragments where a single curve may be described by a number of parameters which enable a quadratic based curve to be rendered in a single output space without the need to perform multiplications. Further data formats such as cubic splines and the like may also be used. An object may contain a mixture of many different edge types. Typically, common to all formats are identifiers for the start and end of each line (whether straight or curved) and typically, these are identified by a scan line number thus defining a specific output space in which the curve may be rendered.

For example, Fig. 16A shows a prior art edge description of an edge 600 that is required to be divided into two segments 601 and 602 in order for the segments to be adequately described and rendered. This arises because the prior art edge description, whilst being simply calculated through a quadratic expression, could not accommodate an inflexion point 604. Thus the edge 600 was dealt with as two separate edges having end points 603 and 604, and 604 and 605 respectively. Fig. 16B shows a cubic spline 610 which is described by end points 611 and 612, and control points 613 and 614. This format requires calculation of a cubic polynomial for render purposes and thus is expensive of computational time.

Figs. 16C and 16D show examples of edges applicable to the preferred embodiment. In the preferred embodiment, a edge is considered as a single entity and if necessary, is partitioned to delineate sections of the edge that may be described in different formats, a specific goal of which is to ensure a minimum level of complexity for the description of each section.

In Fig. 16C, a single edge 620 is illustrated spanning between scan lines A and M. An edge is described by a number of parameters including start_x , start_y , one or more segment descriptions which include an address that points to the next segment in the edge, and a finish segment used to terminate the edge. According to the preferred embodiment, the edge 620 may be described as having three step segments, a vector segment, and a quadratic segment. A step segment is simply defined as having an x-step value and a y-step value. For the three step segments illustrated, the segment descriptions are [0,2], [+2,2], and [+2,0]. Note that the x-step value is signed thereby indicating the direction of the step, whilst the y-step value is unsigned as such is always in a raster scan direction of increasing scan line value. The next segment is a vector segment which typically requires parameters start_x, start_y, finish_y and slope (DX). In this example, because the vector segment is an intermediate segment of the edge 620, the start_x and start_y may be omitted because such arise from the preceding segment(s). The slope value (DX) is signed and is added to the x-value of a preceding scan line to give the x-value of the current scan line, and in the illustrated case, DX = +1. The next segment is a quadratic segment which has a structure corresponding to that of the vector segment, but also a second order value (DDX) which is also signed and is added to DX to alter the slope of the segment.

Fig. 16D shows an example of a cubic curve according the preferred embodiment which includes a description corresponding to the quadratic segment save for the addition of a signed third-order value (DDDX), which is added to DDX to vary the rate of change of slope of the segment. Many other orders may also be implemented.

It will be apparent from the above that the ability to handle plural data formats describing edge segments allows for simplification of edge descriptions and evaluation, without reliance on complex and computationally expensive mathematical operations. In contrast, in the prior art system of Fig. 16A, all edges, whether, orthogonal, vector or quadratic, were required to be described by the quadratic form.

The operation of the preferred embodiment will be described with reference to the simple example of rendering an image 78 shown in Fig. 8. The image 78 is seen to

include two graphical objects, in particular, a partly transparent blue-coloured triangle 80 rendered on top of and thereby partly obscuring an opaque red coloured rectangle 90. As seen, the rectangle 90 includes side edges 92, 94, 96 and 98 defined between various pixel positions (X) and scan line positions (Y). Because the edges 96 and 98 are formed upon the scan lines (and thus parallel therewith), the actual object description of the rectangle 90 can be based solely upon the side edges 92 and 94, such as seen in Fig. 9A. In this connection, edge 92 commences at pixel location (40,35) and extends in a raster direction down the screen to terminate at pixel position (40,105). Similarly, the edge 94 extends from pixel position (160,35) to position (160,105). The horizontal portions of the rectangular graphic object 90 may be obtained merely by scanning from the edge 92 to the edge 94 in a rasterised fashion.

The blue triangular object 80 however is defined by three object edges 82, 84 and 86, each seen as vectors that define the vertices of the triangle. Edges 82 and 84 are seen to commence at pixel location (100,20) and extend respectively to pixel locations (170,90) and (30,90). Edge 86 extends between those two pixel locations in a traditional rasterised direction of left to right. In this specific example because the edge 86 is horizontal like the edges 96 and 98 mentioned above, is it not essential that the edge 86 be defined, since the edge 86 is characterised by the related endpoints of the edges 82 and 84. In addition to the starting and ending pixel locations used to describe the edges 82 and 84, each of these edges will have associated therewith the slope value in this case +1 and -1 respectively.

Fig. 10 shows the manner in which the rectangle 90 is rendered, this commencing on scan line 35 and how the edges 82 and 84 intersect the scan line 35. It will be apparent from Fig. 10 that the rasterisation of the image 78 requires resolution of the two objects 90 and 80 in such a fashion that the object having the higher priority level is rendered "above" that with a lower priority level. This is seen from Fig. 11 which represents an edge list record used for the rendering of the image 78. The record of Fig. 11 includes two entries, one for each of the objects, and which are arranged at a scan line value corresponding to the start, in a raster rendering order, of the respective object.

It will be seen from Fig. 11 that the edge records each have an associated priority level of the object and further detail regarding the nature of the edge being described (eg. colour, slope, etc.)

Returning to Fig. 2, having identified the data necessary to describe the graphic objects to the rendered, the graphic systems 1 then performs a display list generation step 12.

The display list generation 12 is preferably implemented as a software module executing on the host processor 2 with attached ROM 6 and RAM 3. The display list generation 12 converts an object graphics description, expressed in any one or more of the well known graphic description languages, graphic library calls, or any other application specific format, into a display list. The display list is typically written into a display list store 13, generally formed within the RAM 4 but which may alternatively be formed within the rendering stores 30. As seen in Fig. 3, the display list store 13 can include a number of components, one being an instruction stream 14, another being edge information 15 and where appropriate, raster image pixel data 16.

The instruction stream 14 includes code interpretable as instructions to be read by the pixel sequential rendering apparatus 20 to render the specific graphic objects desired in any specific image. For the example of the image shown in Fig. 8, the instruction stream 14 could be of the form of:

(1) render (nothing) to scan line 20;

(2) at scan line 20 add two blue edges 82 and 84;

(3) render to scan line 35;

(4) at scan line 35 add two red edges 92 and 94;

(5) render to completion.

Similarly, the edge information 15 for the example of Fig. 8 may include the following:

edge 84 commences at pixel position 100, edge 82 commences at pixel position 100;

edge 92 commences at pixel position 40, edge 94 commences at pixel position 160;

edge 84 runs for 70 scan lines, edge 82 runs for 70 scan lines;

edge 84 has slope = -1, edge 84 has slope = +1;

edge 92 has slope = 0 edge 94 has slope = 0; and

edges 92 and 94 each run for 70 scan lines.

It will be appreciated from the above example of the instruction stream 14 and edge information 15, and the manner in which each are expressed, that in the image 78 of Fig. 8, the pixel position (X) and the scan line value (Y) define a single output space in which the image 78 is rendered. Other output space configurations however can be realised using the principles of the present disclosure.

Fig. 8 includes no raster image pixel data and hence none need be stored in the store portion 16 of the display list 13, although this feature will be described later.

The display list store 13 is read by a pixel sequential rendering apparatus 20, which is typically implemented as an integrated circuit. The pixel sequential rendering apparatus 20 converts the display list into a stream of raster pixels which can be forwarded to another device, for example, a printer, a display, or a memory store.

Although the preferred embodiment describes the pixel sequential rendering apparatus 20 as an integrated circuit, it may be implemented as an equivalent software module executable on a general purpose processing unit, such as the host processor 2. The software module may form part of a computer program product which may be delivered to a user via a computer readable medium, such as a disk device or computer network.

Fig. 3 shows the configuration of the pixel sequential rendering apparatus 20, the display list store 13 and the temporary rendering stores 30. The processing stages 22 of the pixel-sequential render apparatus 20 include an instruction executor 300, an edge processing module 400, a priority determination module 500, a fill colour determination module 600, a pixel compositing module 700, and a pixel output module 800. The processing operations use the temporary stores 30 which as noted above, may share the

same device (eg. magnetic disk or semiconductor RAM) as the display list store 13, or may be implemented as individual stores for reasons of speed optimisation. The edge processing module 400 uses an edge record store 32 to hold edge information which is carried forward from scan-line to scan-line. The priority determination module 500 uses a priority properties and status table 34 to hold information about each priority, and the current state of each priority with respect to edge crossings while a scan-line is being rendered. The fill colour determination module 600 uses a fill data table 36 to hold information required to determine the fill colour of a particular priority at a particular position. The pixel compositing module 700 uses a pixel compositing stack 38 to hold intermediate results during the determination of an output pixel that requires the colours from multiple priorities to determine its value.

The display list store 13 and the other stores 32-38 detailed above may be implemented in RAM or any other data storage technology.

The processing steps shown in the embodiment of Fig. 3 take the form of a processing pipeline 22. In this case, the modules of the pipeline may execute simultaneously on different portions of image data in parallel, with messages passed between them as described below. In another embodiment, each message described below may take the form of a synchronous transfer of control to a downstream module, with upstream processing suspended until the downstream module completes the processing of the message.

The instruction executor 300 reads and processes instructions from the instruction stream 14 and formats the instructions into messages that transferred via an output 398 to the other modules 400, 500, 600 and 700 within the pipeline 22. In the preferred embodiment, the instruction stream 14 may include the instructions:

LOAD_PRIORITY_PROPERTIES: This instruction is associated with data to be loaded into the priority properties and status table 34, and an address in that table to which the data is to be loaded. When this instruction is encountered by the instruction executor 300, the instruction executor 300 issues a message for the storage of the data in the specified location of the priority properties and status table 34. This may be

accomplished by formatting a message containing this data and passing it down the processing pipeline 22 to the priority determination module 500 which performs the store operation.

LOAD_FILL_DATA: This instruction is associated with data to be loaded into the fill data table 36, and an address in that table to which the data is to be loaded. When this instruction is encountered by the instruction executor 300, the instruction executor 300 issues a message for the storage of the data at the specified address of the fill data table 36. This may be accomplished by formatting a message containing this data and passing it down the processing pipeline 22 to the fill data determination module which performs the store operation.

LOAD_NEW_EDGES_AND_RENDER: This instruction is associated with an address in the display list store 13 of new edges 15 which are to be introduced into the rendering process when a next scan line is rendered. When this instruction is encountered by the instruction executor, the instruction executor 300 formats a message containing this data and passes it to the edge processing module 400. The edge processing module 400 stores the address of the new edges in the edge record store 32. The edges at the specified address are sorted on their initial scan line intersection coordinate before the next scan line is rendered. In one embodiment, the edges are sorted by the display list generation process 12. In another embodiment, the edges are sorted by the pixel-sequential rendering apparatus 20.

SET_SCAN LINE_LENGTH: This instruction is associated with a number of pixels which are to be produced in each rendered scan line. When this instruction is encountered by the instruction executor 300, the instruction executor 300 passes the value to the edge processing module 400 and the pixel compositing module 700.

SET_OPACITY_MODE: This instruction is associated with a flag which indicates whether pixel compositing operations will use an opacity channel (also known in the art as an alpha channel). When this instruction is encountered by the instruction executor 300, the instruction executor 300 passes the flag value in the pixel compositing module 700.

The instruction executor 300 is typically formed by a microcode state machine which maps instructions and decodes them into pipeline operations for passing to the various modules. A corresponding software process may alternatively be used.

The operation of the edge processing module 400 during a scan line render operation will now be described with reference to Fig. 4. The initial conditions for the rendering of a scan line is the availability of three lists of edge records. Any or all of these lists may be empty. These lists are a new edge list 402, obtained from the edge information 15 and which contains new edges as set by the LOAD_NEW_EDGES_AND_RENDER instruction, a main edge list 404 which contains edge records carried forward from the previous scan line, and a spill edge list 406 which also contains edge records carried forward from the previous scan line. Each edge record may include:

(i) a current scan line intersection coordinate (referred to here as the X coordinate),

(ii) a count (referred to herein as NY) of how many scan lines a current segment of this edge will last for (in some embodiments this may be represented as a Y limit),

(iii) a value to be added to the X coordinate of this edge record after each scan line (referred to here as the DX),

(iv) a value to be added to the DX of this edge record after each scan line (referred to here as the DDX),

(v) one or more priority numbers (P),

(vi) a direction (DIR) flag which indicates whether the edge crosses scan lines in an upward (+) or a downward (-) manner, and

(vii) an address (ADD) of a next edge segment in the list.

Such a format accommodates vectors, orthogonally arranged edges and quadratic curves. The addition of further parameters, DDDX for example, may allow such an arrangement to accommodate cubic curves. In some applications, such as cubic Bezier spline, a 6-order polynomial (ie: up to DDDDDDX) may be required.

For the example of the edges 84 and 94 of Fig. 8, the corresponding edge records at scan line 20 could read as follows in Table 1:

TABLE 1

Edge 84	Edge 92
X = 100	X = 40
NY = 70	NY = 70
DX = 1	DX = 0
DDX = 0	DDX = 0
P = 1	P = 0
DIR = (-)	DIR = (+)
ADD = (irrelevant in this example)	ADD = (irrelevant in this example)

In this description, coordinates which step from pixel to pixel along a scan line being generated by the rendering process will be referred to as X coordinates, and coordinates which step from scan line to scan line will be referred to as Y coordinates. Preferably, each edge list contains zero or more records placed contiguously in memory. Other storage arrangements, including the use of pointer chains, are also possible. The records in each of the three lists 402, 404 and 406 are arranged in order of scan line intersection (X) coordinate. This is typically obtained by a sorting process, initially managed by an edge input module 408 which receives messages, including edge information, from the instruction executor 300. It is possible to relax the sort to only regard the integral portion of each scan line intersection coordinate as significant. It is also possible to relax the sort further by only regarding each scan line intersection coordinate, clamped to the minimum and maximum X coordinates which are being produced by the current rendering process. Where appropriate, the edge input module 408 relays messages to modules 500, 600 and 700 downstream in the pipeline 22 via an output 498.

The edge input module 408 maintains references into and receives edge data from each of the three lists 402, 404, and 406. Each of these references is initialised to refer to

the first edge in each list at the start of processing of a scan line. Thereafter, the edge input module 408 selects an edge record from one of the three referenced edge records such that the record selected is the one with the least X coordinate out of the three referenced records. If two or more of the X-records are equal, each are processed in any order and the corresponding edge crossings output in the following fashion. The reference which was used to select that record is then advanced to the next record in that list. The edge just selected is formatted into a message and sent to an edge update module 410. Also, certain fields of the edge, in particular the current X, the priority numbers, and the direction flag, are formatted into a message which is forwarded to the priority determination module 500 as an output 498 of the edge processing module 400. Embodiments which use more or fewer lists than those described here are also possible.

Upon receipt of an edge, the edge update module 410 decrements the count of how many scan lines for which a current segment will last. If that count has reached zero, a new segment is read from the address indicated by the next segment address. A segment specifies:

(i) a value to add to the current X coordinate immediately the segment is read,

(ii) a new DX value for the edge,

(iii)a new DDX value for the edge, and

(iv)a new count of how many scan lines for which the new segment will last.

If there is no next segment available at the indicated address, no further processing is performed on that edge. Otherwise, the edge update module 410 calculates the X coordinate for the next scan line for the edge. This typically would involve taking the current X coordinate and adding to it the DX value. The DX may have the DDX value added to it, as appropriate for the type of edge being handled. The edge is then written into any available free slot in an edge pool 412, which is an array of two or more edge records. If there is no free slot, the edge update module 410 waits for a slot to become available. Once the edge record is written into the edge pool 412, the edge update module 410 signals via a line 416 to an edge output module 414 that a new edge has been added to the edge pool 412.

As an initial condition for the rendering of a scan line, the edge output module 414 has references to each of a next main edge list 420 and a next spill edge list 422, not seen in Fig. 4 but associated with the lists 404 and 406 in the edge record 32. Each of these references is initialised to the location where the, initially empty, lists 420 and 422 may be built up. Upon receipt of the signal 416 indicating that an edge has been added to the edge pool 412, the edge output module 414 determines whether or not the edge just added has a lesser X coordinate than the edge last written to the next main edge list 420 (if any). If this is true, a "spill" is said to have occurred because the edge cannot be appended to the main edge list 404 without violating its ordering criteria. When a spill occurs, the edge is inserted into the next spill edge list 422, preferably in a manner that maintains a sorted next spill edge list 422. For example this may be achieve using a software sorting routine. In some embodiments spills may be triggered by other conditions, such as excessively large X coordinates.

If the edge added to the edge pool 412 has an X coordinate greater than or equal to the edge last written to the next main edge list 420 (if any), and there are no free slots available in the edge pool 412, the edge output module 414 selects the edge from the edge pool 412 which has the least X coordinate, and appends that edge to the next main edge list 420, extending it in the process. The slot in the edge pool 412 which was occupied by that edge is then marked as free.

Once the edge input module 408 has read and forwarded all edges from all three of its input lists 402, 404 and 406, it formats a message which indicates that the end of scan line has been reached and sends the message to both the priority determination module 500 and the edge update module 410. Upon receipt of that message, the edge update module 410 waits for any processing it is currently performing to complete, then forwards the message to the edge output module 414. Upon receipt of the message, the edge output module 414 writes all remaining edge records from the edge pool 412 to the next main edge list 404 in X order. Then, the reference to the next main edge list 420 and the main edge list 404 are exchanged between the edge input module 408 and the edge output module 414, and a similar exchange is performed for the next spill edge list 422 and the

spill edge list 406. In this way the initial conditions for the following scan linecare established.

Rather than sorting the next spill edge list 422 upon insertion of edge records thereto, such edge records may be merely appended to the list 422, and the list 422 sorted at the end of the scan line and before the exchange to the current spill list 406 becomes active in edge rasterisation of the next scan line. Other methods of sorting the edges involving fewer or more lists may be used, as well as different sorting algorithms.

It can be deduced from the above that edge crossing messages are sent to the priority determination module 500 in scan line and pixel order (that is, they are ordered firstly on Y and then on X) and that each edge crossing message is labelled with the priority to which it applies.

Fig. 12A depicts a specific structure of an active edge record 418 that may be created by the edge processing module 400 when a segment of an edge is received. If the first segment of the edge is a step (orthogonal) segment, the X-value of the edge is added to a variable called "X-step" for the first segment to obtain the X position of the activated edge. Otherwise, the X-value of the edge is used. This means that the edges in the new edge record must be sorted by $X_{edge} + X_{step}$. The X_{step} of the first segment should, therefore, be zero, in order to simplify sorting the edges. The Y-value of the first segment is loaded into the NY field of the active edge record 418. The DX field of the active edges copied from the DX field identifier of vector or quadratic segments, and is set to zero for a step segment. A u-flag as seen in Fig. 12A is set if the segment is upwards heading (see the description relating to Fig. 13A). A q-flag is set if the segment is a quadratic segment, and cleared otherwise. An i-flag is provided and is set if the segment is invisible. A d-flag is set when the edge is used as a direct clipping object, without an associated clipping level, and is applicable to closed curves. The actual priority level of the segment, or a level address is copied from the corresponding field of the new edge record into a level (ADDR) field in the active edge record 418. A segment address/DDX field of the active edge record 418 is either the address of the next segment in the segment list or copied from the segment's DDX value, if the segment is quadratic. The segment

address is used to terminate an edge record. As a consequence, in the preferred embodiment, any quadratic curve (ie: that uses the DDX filed) will be a terminal segment of an edge record.

It will be appreciated from Fig. 12A that other data structures are also possible, and necessary for example where higher-order polynomial implementations are used. Further, the segment address and the DDX field may be separated into different fields, and additional flags provided to meet alternate implementations.

Fig. 12B depicts the arrangement of the edge records described above in the preferred embodiment and used in the edge processing module 400. The edge pool 412 is supplemented by a new active edge record 428, a current active edge record 430 and a spill active edge record 432. As seen in Fig. 12B, the records 402, 404, 406, 420 and 422 are dynamically variable in size depending upon the number of edges being rendered at any one time. Each record includes a limit value which, for the case of the new edge list 402, is determined by a SIZE value incorporated with the LOAD_EDGES_AND_RENDER instruction. When such an instruction is encountered, SIZE is checked and if non-zero, the address of the new edge record is loaded and a limit value is calculated which determines a limiting size for the list 402.

Although the preferred embodiments utilises arrays and associated pointers for the handling of edge records, other implementations, such as linked lists for example may be used. These other implementations may be hardware or software-based, or combinations thereof.

The specific rendering of the image 78 shown in Fig. 8 will now be described with reference to scan lines 34, 35 and 36 shown in Fig. 10. In this example, the calculation of the new X co-ordinate for the next scan line is omitted for the purposes of clarity, with Figs. 12C to 12I illustrating the output edge crossing being derived from one of the registers 428, 430 and 432 of the edge pool 412

Fig. 12C illustrates the state of the lists noted above at the end of rendering scan line 34 (the top portion of the semi-transparent blue triangle 80). Note that in scan line 34 there are no new edges and hence the list 402 is empty. Each of the main edge lists 404

and next main edge list 420 include only the edges 82 and 84. Each of the lists includes a corresponding pointer 434, 436, and 440 which, on completion of scan line 34, point to the next vacant record in the corresponding list. Each list also includes a limit pointer 450, denoted by an asterisk (*) which is required to point to the end of the corresponding list. If linked lists were used, such would not be required as linked lists include null pointer terminators that perform a corresponding function.

As noted above, at the commencement of each scan line, the next main edge list 420 and the main edge list 404 are swapped and new edges are received into the new edge list 402. The remaining lists are cleared and each of the pointers set to the first member of each list. For the commencement of scan line 35, the arrangement then appears as seen in Fig. 12D. As is apparent from Fig. 12D, the records include four active edges which, from Fig. 10, are seen to correspond to the edges 92, 94, 84 and 82.

Referring now to Fig. 12E, when rendering starts, the first segment of the new edge record 402 is loaded into an active edge record 428 and the first active edge records of the main edge list 404 and spill edge list 406 are copied to records 430 and 432 respectively. In this example, the spill edge list 406 is empty and hence no loading takes place. The X-positions of the edges within the records 428, 430 and 432 are then compared and an edge crossing is emitted for the edge with the smallest X-position. In this case, the emitted edge is that corresponding to the edge 92 which is output together with its priority value. The pointers 434, 436 and 438 are then updated to point to the next record in the list.

The edge for which the edge crossing was emitted is then updated (in this case by adding DX = 0 to its position), and buffered to the edge pool 412 which, in this example, is sized to retain three edge records. The next entry in the list from which the emitted edge arose (in this case list 402) is loaded into the corresponding record (in this case record 428). This is seen in Fig. 12F.

Further, as is apparent from Fig. 12F, a comparison between the registers 428, 430 and 432 again selects the edge with the least X-value which is output as the appropriate next edge crossing (X=85, P=2). Again, the selected output edge is updated and added to

the edge pool 412 and all the appropriate pointers incremented. In this case, the updated value is given by X <- X + DX, which is evaluated as 84 = 85 - 1. Also, as seen, the new edge pointer 434 is moved, in this case, to the end of the new edge list 402.

In Fig. 12G, the next edge identified with the lowest current X-value is again that obtained from the register 430 which is output as an edge crossing (X=115, P=2). Updating of the edge again occurs with the value be added to the edge pool 412 as shown. At this time, it is seen that the edge pool 412 is now full and from which the edge with the smallest X-value is selected and emitted to the output list 420, and the corresponding limited pointer moved accordingly.

As seen in Fig. 12H, the next lowest edge crossing is that from the register 428 which is output (X=160 P=1). The edge pool 412 is again updated and the next small X-value emitted to the output list 420.

At the end of scan line 35, and as seen in Fig. 12I, the contents of the edge pool 412 are flushed to the output list 420 in order of smallest X-value. As seen in Fig. 12J, the next main edge list 420 and the main edge list 404 are swapped by exchanging their pointers in anticipation of rendering the next scan line 36. After the swapping, it is seen from Fig. 12J that the contents of the main edge list 404 include all edge current on scan line 36 arranged in order of X-position thereby permitting their convenient access which facilitates fast rendering.

Ordinarily, new edges are received by the edge processing module 400 in order of increasing X-position. When a new edge arrives, its position is updated (calculated for the next scan line to be rendered) and this determines further action as follows:

(a) if the updated position is less than the last X-position output on the line 498, the new edge is insertion sorted into the main spill list 406 and the corresponding limit register updated;

(b) otherwise, if there is space, it is retained in the edge pool 412.

As is apparent from the forgoing, the edge pool 412 aids in the updating of the lists in an ordered manner in anticipation of rendering the next scan line in the rasterised image. Further, the size of the edge pool 412 may be varied to accommodate larger

numbers of non-ordered edges. However, it will be appreciated that in practice the edge pool 412 will have a practical limit, generally dependent upon processing speed and available memory with the graphic processing system. In a limiting sense, the edge pool 412 may be omitted which would ordinarily require the updated edges to be insertion sorted into the next output edge list 420. However, in the preferred embodiment this situation is avoided, as a normal occurrence through the use of the spill lists mentioned above. The provision of the spill lists allows the preferred embodiment to be implemented with an edge pool of practical size and yet handle relatively complex edge intersections without having to resort to software intensive sorting procedures. In those small number of cases where the edge pool and spill list are together insufficient to accommodate the edge intersection complexity, sorting methods may be used.

An example of where the spill list procedure is utilised is seen in Fig. 14A where three arbitrary edges 60, 61 and 63 intersect an arbitrary edge 62 at a relative position between scan lines A and B. Further, the actual displayed pixel locations 64 for each of scan lines A, B, are shown which span pixel locations C to J. In the above described example where the edge pool 412 is size to retain three edge records, it will be apparent that such an arrangement alone will not be sufficient to accommodate three edge intersections occurring between adjacent scan lines as illustrated in Fig. 14A.

Fig. 14B shows the state of the edge records after rendering the edges 60, 61 and 63 on scan line. The edge crossing H is that most recently emitted and the edge pool 412 is full with the updated X-values E, G and I for the edges 60, 61 and 63 respectively for the next scan line, scan line B. The edge 62 is loaded into the current active edge record 430 and because the edge pool 412 is full, the lowest X-value, corresponding to the edge 60 is output to the output edge list 420.

In Fig. 14C, the next edge crossing is emitted (X = J for edge 62) and the corresponding updated value determined, in this case X = C for scan line B. Because the new updated value X = C is less than the most recent value X = E copied to the output list 420, the current edge record and its corresponding new updated value is transferred directly to the output spill list 422.

Fig. 14D shows the state of the edge records at the start of scan line B where it is seen that the main and output lists, and their corresponding spill components have been swapped. To determine the first emitted edge, the edge 60 is loaded into the current active edge register 430 and the edge 62 is loaded into the spill active edge register 432. The X-values are compared and the edge 62 with the least X-value (X = C) is emitted, updated and loaded to the edge pool 412.

Edge emission and updating continues for the remaining edges in the main edge list 404 and at the end of the scan line, the edge pool 412 is flushed to reveal the situation shown in Fig. 14E, where it is seen that each of the edges 60 to 63 are appropriately ordered for rendering on the next scan line, having been correctly emitted and rendered on scan line B.

As will be apparent form the forgoing, the spill lists provide for maintaining edge rasterisation order in the presence of complex edge crossing situations. Further, by virtue of the lists being dynamically variable in size, large changes in edge intersection numbers and complexity may be handled without the need to resort to sorting procedures in all but exceptionally complex edge intersections.

In the preferred embodiment the edge pool 412 is sized to retain eight edge records and size of the lists 404, 420 together with their associated spill lists 406, 422, is dynamically variable thereby providing sufficient scope for handling large images with complex edge crossing requirements.

The operation of the priority determination module 50 will now be described with reference to Fig. 5. Incoming messages 498 from the edge processing module 400, which may include set priority data messages, set fill data messages, edge crossing messages, and end of scan line messages, first pass through a first-in first-out (FIFO) buffer 518 before being read by a priority update module 506. The FIFO 518 acts to de-couple the operation of the edge processing module 400 and the priority determination module 500. A priority state table 502, comprising part of the tables 34 mentioned above, is used to hold information about each object priority. Preferably the FIFO 518 is sized to enable the receipt from the edge processing module 400 and transfer to the priority state

table 502 of a full scan line of edge-crossings in a single action. Such permits the priority determination module 500 to efficiently handle multiple edge-crossings at the same pixel (X) location. Each record in the priority state table 502 records:

(i) a fill-rule flag which indicates whether this priority is to have its inside versus outside state determined by the application of the odd-even fill rule or the non-zero winding fill rule;

(ii) a fill count which is modified in a manner indicated by the fill rule each time a edge effecting this priority is crossed;

(iii)a clipper flag which indicates whether this priority is to be used for clipping or filling;

(iv)a clip type flag which, for edges which have the clipper flag set, records whether the clipping type is a "clip-in" or a "clip-out";

(v) a clip count which is decremented and incremented when a clip-in type clip region effecting this priority is entered and exited respectively, and incremented and decremented when a clip-out type clip region effecting this priority is entered and exited respectively; and

(vi)a flag which records whether this priority requires levels beneath it to be calculated first, referred to as the "need-below" flag.

Clipping objects are known in the art and act not to display a particular new object, but rather to modify the shape of an another object in the image. Clipping objects can also be turned-on and turned-off to achieve a variety of visual effects. For example, the object 80 of Fig. 8 could be configured as a clipping object acting upon the object 90 to remove that portion of the object 90 that lies beneath the clipping object 80. This may have the effect of revealing any object or image beneath the object 90 and within the clipping boundaries that would otherwise be obscured by the opacity of the object 90.

Figs. 13A and 13B demonstrate the application of the odd-even and non-zero winding rules. For the purposes of the non-zero winding rule, Fig. 13A illustrates how the edges 71 and 72 of an object 70 are allocated a notional direction, according to whether the edges are downwards-heading or upwards-heading respectively. In order to

form a closed boundary, edges link nose-to-tail around the boundary. The direction given to an edge for the purposes of the fill-rule (applied and described later) is independent of the order in which the segments are defined. Edge segments are defined in the order in which they are tracked, corresponding to the rendering direction.

Fig. 13B shows a single object (a pentagram) having two downwards-heading edges 73 and 76, and three upwards-heading edges 74, 75 and 77. The odd-even rule operates by simply toggling a Boolean value as each edge is crosses the scan line in question, thus effectively turning-on or turning-off an object colour. The non-zero winding rule increments and decrements a fill count value dependent upon the direction of an edge being crossed. In Fig. 13B, the first two edges 73 and 76 encountered at the scan line are downwards-heading and thus traversal of those edge increment the fill count, to +1 and +2 respectively. The next two edges 74 and 77 encountered by the scan line are upwards-heading and accordingly decrement the fill count, to +1 and 0 respectively.

In some embodiments some of this information is associated with edges in the display list 13 and various edge lists described above, and forwarded as part of the edge crossing message to the priority determination module 500. In particular, the fill-rule flag, the clipper flag, the clip type flag, and the need-below flag may be handled in this manner.

Returning to Fig. 5, the priority update module 506 maintains a counter 524 which records the scan line intersection coordinate up to which it has completed processing. This will be referred to as the current X of the priority update module 506. The initial value at the start of a scan line is zero.

Upon examining an edge crossing message received at the head of the FIFO 518, the priority update module 506 compares the X intersection value in the edge crossing message with its current X. If the X intersection value in the edge crossing message is less than or equal to the current X of the priority update module 506 processes the edge crossing message. Edge crossing message processing comes in two forms, "normal edge processing" (described below) is used when the record in the priority state table 502 indicated by the first priority in the edge crossing message has a clipper flag which

indicates that this is not a clip priority, otherwise "clip edge processing" (described below) is performed.

A priority is active at a pixel if the pixel is inside the boundary edges which apply to the priority, according to the fill-rule for that priority, and the clip count for the priority is zero. A priority is exposed if it is the uppermost active priority, or if all the active priorities above it have their corresponding need-below flags set. In this fashion, pixel values may generated using only the fill data of the exposed priorities.

The need-below flag for a priority is established in the information of the display list and is used to inform the pixel generating system that any active priorities beneath the priority in question do not contribute to the pixel value being rendered, unless the flag is set. The flag is cleared where appropriate to prevent extra compositing operations which would otherwise contribute nothing to the final pixel value.

"Normal edge processing" includes, for each priority in the edge crossing message and with reference to fields of the priority state table record indicated by that priority, the steps of:

(i) noting the current fill count of the current priority;

(ii) either:

(a) if the fill rule of the current priority is odd-even, setting the fill count to zero if it is currently non-zero, else setting it to any non-zero value, or

(b) if the fill rule of the current priority is non-zero winding, incrementing or decrementing (depending on the edge direction flag) the fill count; and

(iii) comparing the new fill count with the noted fill count and if one is zero and the other is non-zero performing an "active flag update" (described below) operation on the current priority.

Some embodiments may use a separate edge crossing message for each priority rather than placing a plurality of priorities in each edge crossing message.

An active flag update operation includes first establishing a new active flag for the current priority. The active flag is non-zero if the fill count for the priority in the priority state table 502 is non-zero and the clip count for the priority is zero, else the active flag is

zero. The second step in the active flag update operation is to store the determined active flag in an active flags array 508 at the position indicated by the current priority, then if the need-below flag in the priority state table for the current priority is zero, also storing the active flag in an opaque active flags array 510 at the position indicated by the current priority.

"Clip edge processing" includes, with reference to fields of the priority state table record indicated by the first priority in the edge crossing message, the steps of:

(i) noting the current fill count of the current priority;

(ii) either:

(a) if the fill rule of the current priority is odd-even, setting the fill count to zero if it is currently non-zero else setting it to any non-zero value, or

(b) if the fill rule of the current priority is non-zero winding, incrementing or decrementing (depending on the edge direction flag) the fill count; and

(iii) comparing the new fill count with the noted fill count and determining a clip delta value of:

(a) zero, if both the new fill count is zero and the noted fill count is zero, or both the new fill count is non-zero and the noted fill count is non-zero,

(b) plus one, if the clip type flag of the current priority is clip-out and the noted fill count is zero and the new fill count is non-zero, or the clip type flag of the current priority is clip-in and the noted fill count is non-zero and the new fill count is zero, or otherwise,

(c) minus one; and

(iv) for every subsequent priority after the first in the edge crossing message, add the determined clip delta value to the clip count in the record in the priority state stable indicated by that subsequent priority, and if the clip count either moved from non-zero to zero, or from zero to non-zero in that process, performing an active flag update operation as described above on that subsequent priority. It should be noted that the initial value of each clip count is set by the LOAD_LEVEL_PROPERTIES

instruction described previously. The clip count is typically initialised to the number of clip-in priorities which affect each priority.

Some embodiments do not associate a priority with a clip, but instead directly increment and decrement the clip count of all priorities given in the edge crossing message. This technique can be used, for example, when clip shapes are simple and do not require the application of a complex fill rule. In this specific application, the clip count of the level controlled by an edge is incremented for an upwards heading edge or decremented for a downwards heading edge. A simple closed curve, described anticlockwise, acts a clip-in, whereas a simple closed curve, described clockwise, acts as a clip-out.

When the X intersection value in the edge crossing message is greater than the current X of the priority update module 506, the priority update module 506 forms a count of how many pixels to generate, being the difference between the X intersection value in the edge crossing message and the current X, this count is formatted into a priority generation message, which is sent via a connection 520 to a priority generation module 516. The priority update module 506 then waits for a signal 522 from the priority generation module 516 indicating that processing for the given number of pixels has completed. Upon receipt of the signal 522, the priority update module 506 sets its current X to the X intersection value in the edge crossing message and continues processing as described above.

The priority generation module 516 operates with reference to a priority data table 504, also formed within the tables 34, which is used to hold information about each priority. Each record in the priority data table 504 may include:

(i) a fill table address,

(ii) a fill type,

(iii) a raster operation code,

(iv) an alpha channel operation code,

(v) a "source pop" flag,

(vi) a "destination pop" flag, and

(vii) a flag which records whether the colour of this priority is constant for a given Y, referred to here as the "x-independent" flag.

Upon receipt of a priority generation message 520, the priority generation module 516 performs a "pixel priority generation operation" (described below) a number of times indicated by the count it has been supplied, thereupon it signals 522 the priority update module 506 that it has completed the operation.

Each pixel priority generation operation includes firstly using a priority encoder 514 (eg. a 4096 to 12 bit priority encoder) on the opaque active flags array 510 to determine the priority number of the highest opaque active flag. This priority (if any) is used to index the priority data table 504 and the contents of the record so referenced is formed into a fill priority message output 598 from the priority generation module 516 and sent to the fill colour determination module 600. Further, if a priority was determined by the previous step (ie. there was at least one opaque active flag set), the determined priority is held, and is referred to as the "current priority". If no priority was determined the current priority is set to zero. The priority generation module 516 then repeatedly uses a modified priority encoder 512 on the active flag array 508 to determine the lowest active flag which is greater than the current priority. The priority so determined (if any) is used to index the priority data table 504 and the contents of the record so referenced is formed into a fill priority message and is sent 598 to the fill colour determination module 500, then the determined priority is used to update the current priority. This step is used repeatedly until there is no priority determined (that is, there is no priority flagged in the active flags which is greater than the current priority). Then the priority generation module 516 forms an end of pixel message and sends it to the fill colour determination module 600.

As a preferred feature to the basic operation described above, the priority generation module 516 notes the value of the x-independent flag of each message which it forwards to the fill colour determination module 600 while it processes the first pixel of a sequence. If all the forwarded messages have the x-independent flag specified, all subsequent messages in the span of pixels between adjacent edge intersections can be

replaced by a single repeat specification of count-minus one. This is done by producing a repeat message and sending it to the fill colour determination module 600 in place of all further processing in this sequence.

As another preferred feature to the basic operation described above, the priority generation module 516 sends the highest opaque priority via the connection 522 to the priority update module 506 after each level generation message. The priority update module 506 holds this in a store 526. The priority determination module 506 then, instead of a simple test that the X intersection in the message is greater than the current X, performs a test that the X intersection in the message is greater than the current X and that at least one of the levels in the message is greater than or equal to the highest opaque priority, before producing a pixel priority generation message. By doing this, fewer pixel priority determination operations may be performed and longer repeat sequences may be generated.

Where the repeat message or sequence of operation is not utilized, as may be desired in some implementations, a similar function may be achieved through the incorporation of a cache or FIFO (not illustrated) at the output of the priority generation module 516. Such may be implemented by a four cell cache, for example. The cache allows the priority update module 506 continue working as soon as the cache is loaded, thereby permitting the generation of the next priority level independent of the flushing of the cache to the output 598.

Using the example of the graphic objects shown in Figs. 8 and 9, the priority update process described above can be illustrated, for scan line 35 using the edge crossings seen from Figs. 12C to 12I, as seen in Figs. 15A to 15E.

Figs. 15A to 15E illustrate operation of the priority tables 502 and 504, which in the preferred embodiment are merged for into a single table, called a level activation table 530, together with arrays 508, 510 and encoders 512 and 514. As seen in Fig. 15A, edge crossing messages are received in order for a scan line from the edge processing module 400 and are loaded into the table 530, which is arranged in priority order. The edge crossing messages include, in this example, an incrementing direction according to

the non-zero winging rule of the edge traversal. It is possible for no entries in the priority table 530 to be set.

The level activation table as illustrated 530 includes column entries for fill count, which are determined from the edge according to the non-zero winding rule or, where appropriate, the odd-even rule. The need-below flag is a property of a priority and is set as part of the LOAD_PRIORITIES_PROPERTIES introduction. The need-below is set for all priority levels when the table 530 is loaded. Other columns such as "clip count" and "fill index table" may be used, but for this example are omitted for simplicity of explanation. Where no level is active the corresponding entries are set to zero. Further, the values of the arrays 510 and 508 are updated from the table 530 after receiving a subsequent edge crossing.

From Fig. 15A, it will be apparent that, for convenience, a number of records have been omitted for clarity. Typically, the level activation table 530 would include, arranged in priority order, the following records:

- fill count
- clip count
- fill type
- activation condition and flags, including
    - (i) need - below flag
    - (ii) clip type
    - (iii) clipper flag
- compositing graphics operations and flags, including
    - (i) the raster operation code
    - (ii) the alpha channel operation code
    - (iii) the "source pop" flag
    - (iv) the "destination pop" flag
    - (v) the x - independent flag
- fill rule
- attributes and

- fill table index.

The contents of the table 530, where not used in the priority determination module 500 are passed as messages to each of the fill colour determination module 600 for pixel generation, and to the pixel compositing module 700 for compositing operations.

The first edge crossing for scan line 35 (Fig. 12E) is seen in Fig. 15A where for P=1, the fill count is updated to the value of the edge according to the non-zero winding rule. Because there are no objects beneath, the "need-below" is level set at zero.

Because a previous state of the table 530 was not set, the arrays 510 and 508 remain not set and the priority encoder 514 is disabled from outputting a priority. This is interpreted by priority generation module 516 which outputs a count n=40 (pixels) for a "no object" priority (eg: P = 0), being the first, blank, portion of the scan line 35.

Fig. 15B shows the arrangement when the edge crossing of Fig. 12F is received. The fill count is updated. The arrays 510 and 508 are then set with the previous highest level from the table 530. At this time, the module 516 outputs a count n=45, P=1 representing the edge 96 of the opaque red object 90 before intersection with the semitransparent triangle 80.

Fig. 15C shows the arrangement when the edge crossing of Fig. 12G is received. Note that the fill count has been adjusted downwardly because of the non-zero winding rule. Because the object that is valid prior to receiving the current edge crossing is not opaque, the modified priority encoder 512 is used to select the priority P=2 as the highest active level which is output as is current for n=(115-85)=30 pixels.

Fig. 15D shows the arrangement when the edge crossing of Fig. 12H is received. Note that previously changed "need-below" for P=2 has been transferred to the active array 508, thus permitting the priority encoder to output a value P=1 current for n=(160-115)=45 pixels.

Fig. 15D shows the result when the edge crossing of Fig. 12I is received, providing for an output of P=0 for n=(180-160)=20 pixels.

As such, the priority module 500 outputs counts of pixels and corresponding priority display values for all pixels of a scan line.

The operation of the fill colour determination module 600 will now be described with reference to Fig. 6. Incoming messages 598 from the priority determination module 500, which include set fill data messages, repeat messages, fill priority messages, end of pixel messages, and end of scan line messages, first pass to a fill lookup and control module 604. The fill lookup and control module 604 maintains a current X position counter 614 and a current Y position counter 616 for use by various components of the fill colour determination module 600.

Upon receipt of an end of scan line message, the fill lookup and control module 604 resets the current X counter 614 to zero and increments the current Y counter 616. The end of scan line message is then passed to the pixel compositing module 700.

Upon receipt of a set fill data message, the fill lookup and control module 604 stores the data in the specified location 602 of the fill data table 36.

Upon receipt of a repeat message, the fill lookup and control module 604 increments the current X counter 614 by the count from the repeat message. The repeat message is then passed to the pixel compositing module 700.

Upon receipt of an end of pixel message, the fill lookup and control module 604 again increments the current X counter 614, and the end of pixel message is then passed to the pixel compositing module 700.

Upon receipt of a fill priority message, the fill lookup and control module 604 performs operations which include:

(i) the fill type from the fill priority message is used to select a record size in the table 36;

(ii) the fill table address from the fill priority message, and the record size as determined above, is used to select a record from the fill data table 36;

(iii) the fill type from the fill priority message is used to determine and select a sub-module to perform generation of the fill colour. The sub-modules may include a raster image module 606, a flat colour module 608, a linearly ramped colour module 610, and an opacity tile module 612;

(iv) the determined record is supplied to the selected sub-module 606-612;

(v) the selected sub-module 606-612 uses the supplied data to determine a colour and opacity value;

(vi) the determined colour and opacity is combined with remaining information from the fill colour message, namely the raster operation code, the alpha channel operation code, the source pop flag, and the destination pop flag, to form a colour composite message, which is sent to the pixel compositing module 700 via the connection 698.

In the preferred embodiment the determined colour and opacity is a red, green, blue and opacity quadruple with 8-bit precision in the usual manner giving 32 bits per pixel. However, a cyan, magenta, yellow and black quadruple with an implied opacity, or one of many other known colour representations may alternatively be used. The red, green, blue and opacity case is used in the description below, but the description may also be applied to other cases.

The operation of the raster image module 606, the flat colour module 608, the linearly ramped colour module 610, and the opacity tile module 612 will now be described.

The flat colour module 608 interprets the supplied record as a fixed format record containing three 8-bit colour components (typically interpreted as red, green and blue components) and an 8-bit opacity value (typically interpreted as a measure of the fraction of a pixel which is covered by the specified colour, where 0 means no coverage, that is complete transparency, and 255 means complete coverage, that is, completely opaque). This colour and opacity value is output directly via the connection 698 and forms the determined colour and opacity without further processing.

The linearly ramped colour module 610 interprets the supplied record as a fixed format record containing four sets of constants cx, cy and d, associated with the three colour and one opacity components, and two position values *xbase* and *ybase* being the coordinates of the reference point of the linear ramp. At the reference point, the colour and opacity components have their associated d value.

For each of the four sets, a result value r is computed by combining three constants with the current X and Y coordinates, and the *xbase* and *ybase* constants, using the formula:

$$r = clamp\ (cx \times (X - xbase) + cy \times (Y - ybase) + d)$$

where the function *clamp* is defined as:

$$clamp\ (x) = \begin{cases} 255 & 255 \le x \\ \lfloor x \rfloor & 0 \le x < 255 \\ 0 & x < 0 \end{cases}$$

In an alternative implementation, the linearly ramped colour module 610 interprets the supplied record as a fixed format record containing four sets of three constants, *cx*, *cy*, and *d*, being associated with the three colour and one opacity components. For each of these four sets, a result value *r* is computed by combining the three constants with the current X count, *x*, and the current Y count, *y*, using the formula:

$$r = clamp\ (cx \times x + cy \times y + d)$$

where the function *clamp* is defined as above.

The four results so produced are formed into a colour and opacity value. This colour and opacity value is output directly via the connection 698 and forms the determined colour and opacity without further processing.

Other mathematical calculations giving the same result may be used.

The opacity tile module 612 interprets the supplied record as a fixed format record containing three 8-bit colour components, an 8-bit opacity value, an integer X phase, (*px*), a Y phase, (*py*), an X scale, (*sx*), a Y scale, (*sy*), and a 64 bit mask. These values originate in the display list generation and contained typically in the original page description. A bit address, *a*, in the bit mask, is determined by the formula:

$$a = ((x/2^{sx} + px) \bmod 8) + ((y/2^{sy} + py) \bmod 8) \times 8$$

The bit at the address "*a*" in the bit mask is examined. If the examined bit is one, the colour and opacity from the record is copied directly to the output of the module 612 and forms the determined colour and opacity. If the examined bit is zero, a colour having

three zero component values and a zero opacity value is formed and output as the determined colour and opacity.

The raster image module 606 interprets the supplied record as a fixed format record containing six constants, *a*, *b*, *c*, *d*, *xbase* and *ybase*; an integer count of the number of bits (*bpl*) in each raster line of the raster image pixel data 16 to be sampled; and a pixel type. The pixel type indicates whether the pixel data 16 in the raster image pixel data is to be interpreted as one of:

(i) one bit per pixel black and white opaque pixels;

(ii) one bit per pixel opaque black or transparent pixels;

(iii)8 bits per pixel grey scale opaque pixels;

(iv)8 bits per pixel black opacity scale pixels;

(v) 24 bits per pixel opaque three colour component pixels;, or

(vi)32 bits per pixel three colour component plus opacity pixels.

Many other formats are possible.

The raster image module 606 uses the pixel type indicator to determine a pixel size (*bpp*) in bits. Then a bit address, *a*, in the raster image pixel data 16 is calculated having the formula:

$$a = bpp * \lfloor a \times (x - xbase) + c \times (y - ybase) \rfloor + bpl \times \lfloor b \times (x - xbase) + d \times (y - ybase) \rfloor .$$

A pixel interpreted according to the pixel type from the record 602 is fetched from the calculated address "a" in the raster image pixel data 16. The pixel is expanded as necessary to have three eight bit colour components and an eight bit opacity component. By "expanded", it is meant for example, that a pixel from an eight bit per pixel grey scale opaque raster image would have the sampled eight bit value applied to each of the red, green and blue component, and the opacity component set to fully opaque. This then forms the determined colour and opacity output 698 to the pixel compositing module 700.

As a consequence, the raster pixel data valid within a displayable object is obtained through the determination of a mapping to the pixel image data within the memory 16. This effectively implements an affine transform of the raster pixel data into

the object-based image and is more efficient than prior art methods which transfer pixel data from an image source to a frame store where compositing with graphic object may occur.

As a preferred feature to the above, interpolation between pixels in the raster image pixel data 16 may optionally be performed by first calculating intermediate results p, and q according to the formulae:

$$p = a \times (x - xbase) + c \times (y - ybase)$$

$$q = b \times (x - xbase) + d \times (y - ybase).$$

Next the bit addresses, a_{00}, a_{01}, a_{10}, and a_{11}, of four pixels in the raster image pixel data 16 are determined according to the formulae:

$$a_{00} = bpp \times \lfloor p \rfloor + bpl \times \lfloor q \rfloor$$

$$a_{01} = a_{00} + bpp$$

$$a_{10} = a_{00} + bpl$$

$$a_{11} = a_{00} + bpl + bpp$$

Next, a result pixel component value, r, is determined for each colour and opacity component according to the formula:

$$r = interp\,(\,interp\,(\,get(a_{00}), get(a_{01}), p),\ interp\,(\,get(a_{10}), get(a_{11}), p),\ q\,)$$

where the function *interp* is defined as:

$$interp\,(a, b, c) = a + (b-a) \times (c - \lfloor c \rfloor)$$

In the above equations, the representation $\lfloor value \rfloor = floor\ (value)$, where a *floor* operation involves discarding the fractional part of *value*.

The *get* function returns the value of the current pixel component sampled from the raster image pixel data 16 at the given bit address. Note that for some components of some image types this can an implied value.

As a preferred feature to the above, image tiling may optionally be performed by using x and y values in the above equations which are derived from the current X and Y counters 614, 616 by a modulus operation with a tile size read from the supplied record.

Many more such fill colour generation sub-modules are possible.

The operation of the pixel compositing module 700 will now be described. Incoming messages from the fill colour determination module 600, which include repeat messages, colour composite messages, end of pixel messages, and end of scan line messages are processed in sequence.

Upon receipt of a repeat message or an end of scan line message, the pixel compositing module 700 forwards the message to a pixel output FIFO 702 without further processing.

Upon receipt of a colour composite message the pixel compositing module 700 typically, and in general terms combines the colour and opacity from the colour composite message with a colour and opacity popped from the pixel compositing stack 38 according to the raster operation and alpha channel operation from the colour composite message. It then pushes the result back onto the pixel compositing stack 38. A description of the processing performed upon of receipt of a colour composite message is given below.

Upon receipt of an end of pixel message, the pixel compositing module 700 pops a colour and opacity from the pixel compositing stack 38, with the exception that if the stack 38 is empty an opaque white value is used. The resultant colour and opacity is formed into an pixel output message which is forwarded to the pixel output FIFO.

A known compositing approach is that described in "Compositing Digital Images", Porter, T; Duff, T; Computer Graphics, Vol. 18 No. 3 (1984) pp. 253-259. Examples of Porter and Duff compositing operations are shown in Fig. 21. However, such an approach is deficient in that it only permits handling source and destination colour in the intersecting region formed by the composite and, as a consequence, is unable to accommodate the influence of transparency in the intersecting region. This results in the raster operations defined by Porter and Duff as being essentially inoperative in the presence of transparency.

The processing performed by the pixel compositing module 700 upon receipt of a colour composite message will now be described.

Upon receipt of a colour composite message, the pixel compositing module 700 first forms a *source colour and opacity*. This is taken from the colour and opacity provided in the colour composite message unless the pop source flag is set in the colour composite message, in which case the colour is popped from the pixel compositing stack 38 instead. If at this time, or during any pop of the pixel compositing stack, the pixel compositing stack 38 is found to be empty, an opaque white colour value is used without any error indication. Next, a *destination colour and opacity* is popped from the pixel compositing stack 38, except that if the destination pop flag is not set in the colour composite message, the stack pointer is not disturbed during the "pop" operation, in effect making this a read from top of stack 38 instead.

The method of combining the source colour and opacity with the destination colour and opacity will now be described with reference to Figs. 7A to 7C. For the purposes of this description, colour and opacity values are considered to range from 0 to 1, (ie: normalised) although they are typically stored as 8-bit values in the range 0 to 255. For the purposes of compositing together two pixels, each pixel is regarded as being divided into two regions, one region being fully opaque and the other fully transparent, with the opacity value being an indication of the proportion of these two regions. Fig. 7A shows a source pixel 702 which has some three component colour value not shown in the figure and an opacity value, (*so*). The shaded region of the source pixel 702 represents the fully opaque portion 704 of the pixel 702. Similarly, the non-shaded region in Fig. 7A represents that proportion 706 of the source pixel 702 considered to be fully transparent. Fig. 7B shows a destination pixel 710 with some opacity value, (*do*). The shaded region of the destination pixel 710 represents the fully opaque portion 712 of the pixel 710. Similarly, the pixel 710 has a fully transparent portion 714. The opaque regions of the source pixel 702 and destination pixel 710 are, for the purposes of the combination, considered to be orthogonal to each other. The overlay 716 of these two pixels is shown in Fig. 7C. Three regions of interest exist, which include a source outside destination 718 which has an area of $so \times (1 - do)$, a source intersect destination 720 which has an area of $so \times do$, and a destination outside

source 722 which has an area of $(1 - so) \times do$. The colour value of each of these three regions is calculated conceptually independently. The source outside destination region 718 takes its colour directly from the source colour. The destination outside source region 722 takes its colour directly from the destination colour. The source intersect destination region 720 takes its colour from a combination of the source and destination colour. The process of combining source and destination colour, as distinct from the other operations discussed above is termed a raster operation and is one of a set of functions as specified by the raster operation code from the pixel composite message. Some of the raster operations included in the preferred embodiment are shown in Table 2.

TABLE 2

Raster operation code	Operation	Comment
0x00	r = 0	BLACKNESS
0x01	r = src & dest	SRCAND
0x02	r = src & ~dest	SRCERASE
0x03	r = src	SRCCOPY
0x04	r = ~src & dest	
0x05	r = dest	NOP
0x06	r = src ^ dest	SRCINVERT
0x07	r = src | dest	SRCPAINT
0x08	r = ~(src | dest)	NOTSRCERASE
0x09	r = ~(src ^ dest)	
0x0a	r = ~dest	DSTINVERT
0x0b	r = src | ~dest	
0x0c	r = ~src	NOTSRCCOPY
0x0d	r = ~src | dest	MERGEPAINT
0x0e	r = ~(src & dest)	
0x0f	r = 0xff	WHITENESS
0x10	r = min(src, dest)	
0x11	r = max(src, dest)	
0x12	r = clamp(src + dest)	
0x13	r = src	
0x14	r = clamp(src - dest)	
0x15	r = dest	
0x16	r = clamp(dest - src)	
0x17	r = clamp(src + dest) where dest is signed	
0x18	r = threshold (dest, src)	
0x19	r = threshold (src, dest)	
0x1a	r = ~ dest	
0x1b	o = luminance (dest, src)	
0x1c	r = ~src	
0x1d	o = ckey(dest; src +/- o)	

Each function is applied to each pair of corresponding colour components of the source and destination colour to obtain a like component in the resultant colour. Many other functions are possible.

The alpha channel operation from the composite pixel message is also considered. The alpha channel operation is performed using three flags, each of which corresponds to one of the regions of interest in the overlay 716 of the source pixel 702 and the destination pixel 710. For each of the regions, a region opacity value is formed which is zero if the corresponding flag in the alpha channel operation is not set, else it is the area of the region.

The resultant opacity is formed from the sum of the region opacities. Each component of the result colour is then formed by the sum of the products of each pair of region colour and region opacity, divided by the resultant opacity.

The resultant colour and opacity is pushed onto the pixel compositing stack 38.

Expression trees are often used to describe the compositing operations required to form an image, and typically comprise a plurality of nodes including leaf nodes, unary nodes and binary nodes. A leaf node is the outermost node of an expression tree, has no descendent nodes and represents a primitive constituent of an image. Unary nodes represent an operation which modifies the pixel data coming out of the part of the tree below the unary operator. A binary node typically branches to left and right subtrees, wherein each subtree is itself is an expression tree comprising at least one leaf node.

When compositing with arbitrary shaped objects, there arises a problem that the various stack operations mentioned above are different for different areas of the image, these depending upon those objects that are active at particular locations.

Figs. 17A and 17B show a typical binary operation (illustrated as an expression tree) between source (S) and destination (D). Regardless of the actual operation being performed, the binary operation of Fig. 17A resolves into four cases or regions of activity as indicated below:

1. (A)S active, (B)D inactive;
2. (A)S active, (B)D active;

3. (A)S inactive, (B)D active; and
4. (A)S inactive, (B)D inactive.

Case 4 always results in no operation (*NOP*) being required to be performed and as a consequence, there exists three different combinations of active levels for a binary tree. Extensions of this concept to tertiary, quaternary and higher order trees will be apparent to those skilled in the art.

As a consequence, when building the compositing stack 38 (for the binary example), one of the three above identified operations is required to be implemented by the stack. Further, the different operations associated with each object in the stack depend upon what is below the object in the level activation table. For rendering of objects using simple OVER operations, as occurs in the Painter's Algorithm, this poses no problem. However for other compositing operations, the stack operations need to be changed depending on the activity of the operands of the compositing operation. While this can be done by clipping the levels providing the stack operations, the number of clips applying to each level can rapidly rise, creating difficulties in handling the stack operations. Examples of problematic operations are the Porter and Duff operations OUT and ROUT as seen in Fig. 21 where an object (operand) clips (alters the boundary of) the other object and has variable transparency in the intersection region.

In order to address this problem, a further table, noted herein as an "activity" table is provided which acts as an adjunct to the level activation table to provide for a logical determination of the alternative actions mentioned above.

Fig. 18A illustrates a generic activity table 800 which includes essentially three sections. A first section 802 provides activation conditions for a specific fill level being processed. The second section 804 includes each of the different actions referred to above as appropriate for the respective levels (specifically for the binary example). The third section 806 indicates whether the source or destination object is active at the particular level. It is noted that the entries contained in the action section 804 may be the specific operations themselves or alternatively pointers to the level activation table where appropriate.

It is also noted that the various operations can provide data to other operations in some cases, but not in others. As a consequence, the activity table 800 can be modified to incorporate flags indicating various conditions for which the operation provides data.

A data structure for a preferred form of the activity table is seen in Fig. 18B as the table 810. The table 810 includes a pointer 814 (*Next Level*) to an entry for the next operation which uses the data, and a flag 806 (or a set of flags where tertiary and higher order trees are being used) that indicates the branch of the expression tree for which the operation is providing data (*Src_Active/Dest_active*). The table 810 also includes a flag 816 that indicates whether the operation is providing data in the source or destination branch. If so, the *Src_active* or *Dest_Active* flags 806 in the next level entry are adjusted accordingly when an activity state 818 of the operation changes. Since an operation only provides data in certain combinations, further flags 812 (*data_in_**) are provided to indicate this. The flags 812, in combinations with the *Src/Dest_Active* flags 806, determine the activity state of a level. Further, since any operation only has to alter the state of the next level if its own activity state changes, the node active flag 818 is provided to monitor such a situation.

For right leaf nodes, it is therefore necessary to activate a *Push* operation and the *Dest_active* flag in the next operation record. For left leaf nodes, it is necessary to activate the *Src_active* flag on an edge crossing, noting that the destination may already be active.

In Fig. 18B, the activation conditions 802 include the fill rule which determines activation of leaf nodes, and the fill count which is used in the manner as described above for the level activation table. The clip count operates also in the manner described above. Edge crossings activate (source) levels in the table 810.

When an activation state of a level changes, the change propagates to the level pointed to by the *Next Level* entry 814. Depending upon the state of the *Data_is_Src* flag 816, the *Src_Active/Dest_Active flag* 806 is changed in the next level. The change propagates if the state of the next level also changes. The table entries contain operations for cases 1, 2 and 3 respectively. These may be pointers to levels within the level

activation table, or actual operations (eg. Alpha operation, colour operations, fill type and stack operation flags). Alternatively, they may be *NULL* if no operation is required.

The activation state of a node level is determined by whether there is data for the next operation in the expression tree, as determined by the *data_in* flags 812 for each of $S \cap \overline{Dop}, S \cap Dop, \overline{S} \cap Dop$ and the *Src / Dest_active* flags 806 for the node level. This is depicted in the table as a *Node_Active* flag 818.

A specific example of this implementation is shown for an expression tree 830 seen in Fig. 19, and with a corresponding initial activity table 820 seen in Fig. 20A.

The expression tree 830 provides for the rendering of operands *A*, *B* and *C* onto a page, where the latter, for the sake of completeness is seen in the tree 830 as a right leaf node, *PAGE*. *PAGE* is always active and encompasses the entire image output space and therefore it may be omitted from the activity table 820.

Since *B* and C are leaf nodes, these form the lower levels of the table 820 and each result in activation operations 804 that can cause a push of the respective operator onto the compositing stack. Since each are right leaf nodes, *C* is pushed first, and the $\overline{S} \cap Dop$ is a *NOP* because nothing operates on operand C. The *data_in_*op* flags 812, *Next Level* 814 and *Data_is_Src* 816 flags are also updated. Operand B results in corresponding actions.

The next level in the activity table 820 is formed by the left leaf node *A*, and its corresponding operator *Op2*. Activation operations 804 for this level are updated with each of $\overline{S} \cap Dop$ and $S \cap Dop$ being *Aop2B* each modified by a qualifier *a* or *b* respectively depicting the differences between the operations. The operation *Op2* only provides data in *S*, and this is represented by an activity which pops *B* off the stack if *D* is active and *S* is not (ie. $\overline{S} \cap Dop$).

The next level in the table 820 relates to *Op1* and produces respective qualified results *a*, *b* and *c* in the activation operations 804. For the final level *over*, since *PAGE* is always active, $S \cap \overline{Dop}$ and $\overline{S} \cap Dop$ result in *NOP* to be pushed onto the stack. Only in the simple intersection $S \cap Dop$, is *over* active.

For this example, let us consider what happens if *A*, *B* and *C* are initially inactive, and those operands are subsequently activated in turn.

If *A* is activated first, then *AOp2a* is activated on this level, reflected by the setting of the *Src_Active* flag 806. Since the $S \cap \overline{D}$ flag 812 is set (since B is not yet active), the *Node_Active* flag 818 is then set for the level. Since the state of *Node_Active* has changed, the *Src_Active* flag 806 in the *Op1* level is set. Note that *Data_is_Src* 816 is set for *AOp2* level. The *Op1* level has *Src_Active* and $\overline{Dest\_Active}$ (since C is yet to be activated) so *Op1a* is activated on this level. Since $S \cap \overline{D}$ is set, *Node_Active* 818 is then set for the *Op1* level. Since the state of *Node_Active* 818 has changed, the *Src_Active* flag 806 in the *over* level is set. Since the *over* level has *Src_active* and *Dest_Active* (because *PAGE* is always active), then *over* is activated on this level. Since $S \cap D$ is set, *Node_Active* is set and the state of *Node_Active* has not changed. No further action is then required. The compositing stack 38, at this stage may be established from the table 820 to appear as seen in Fig. 20B.

Turning now to Fig. 20C, if *B* is activated next, *Push B* is activated on this level since $S \cap \overline{D}$ is set (and *D* is irrelevant, anyway). *Node_Active* 818 is set for this level. The state of *Node_Active* has changed, and therefore *Dest_Active* flag in the *AOp2* level is set. *AOp2bB* is then activated, and *AOp2aB* is deactivated. Since $S \cap D$ is set, *Node_Active* remains set and the state of *Node_Active* is unchanged for *AOp2a*. No further action results. The compositing stack 38 then appears as seen in Fig. 20D.

As seen in Fig. 20E, if *C* is activated next, *Push C* is activated on this level. Since *S* is active and *D* is irrelevant, *Node_Active* 818 is set for this level. Since *Node_active* has changed, the *Dest_Active* flag in *Op1* level is then set. *Op1b* is activated so *Op1a* is deactivated. Since data in $S \cap D$ is set, *Node_Active* remains set. Since *Node_Active* is unchanged for the *Op1* level, no further action is required. The compositing stack 38 then appears as seen in Fig. 20F.

This procedure continues for the evaluation of the entire expression tree of Fig. 19, thus providing for the activity table 820 to be established in such a way that various operations are established which may then be pushed onto the compositing stack as

required by the various activation conditions 802 and activity indicators 804. In this fashion, regardless of the type of clipping or other operation being performed, the stack is able to be maintained with the correct operation at the correct level irrespective of the complexity of the expression tree being evaluated. Significantly, whilst the bulk of the expression tree is evaluated through the formation of the display list, generation of the display list is typically unable to account for variations in the operation of various objects such as clipping, these operations being required to implemented during the evaluation of the compositing expression.

It is further noted that further flags, one for *src_is_leaf_node*, which may be activated by edge crossings, and another for *dest_is_PAGE* (always active), can be useful. If *dest_is_PAGE*, it is possible to ignore the $S \sqcap \overline{D}$ case as this never happens.

The above illustrates how the activity table 820 is built based on the structure of the expression tree 830, and has its entries completed (ie. filled in) through the changing activation of the various operands of the tree 830. For the specific example of the table 820, 72 (=2x2x3x3x2) stack structures can arise to account for the different activations and possible outcomes. In this fashion, logical evaluation of the conditions 802, 806, 812, 814, 816 and 818 results in the correct activity 804 being identified as the appropriate stack operation for the particular level.

In an alternative implementation, rather than being constructed as an independent table, the activity table 820 may be merged into the level activation table 530 to give a combined table 830. This avoids the replication of data whilst permitting the priority encoders 512,514 to select not only the correct edge priority but also activation operation, the latter being transferred (progressively) to the pixel compositing stack 38 for evaluation by the compositing module 700 using the fill colours derived from the module 600. such an arrangement is depicted functionally in Fig. 20G.

Alternatively, as seen functionally in Fig. 20H, the activity table 820 may precede the level activation table 530. In such a case, columns for fill count and clip count are included in the activity table 820 and may be omitted from the level activation table 530. In a further alternative configuration, shown functionally in Fig. 20I, the activity

table 820 may follow the level activation table 530. In that case, the activity table 820 can omit fill count and clip count as these are included in the level activation table 530. In some applications, where an activity table 820 is configured as shown in Fig. 20A, the level activation table 530 may be omitted.

The operation codes described above with reference to the activity table 820 and the stack 38 derive from the display list and the instruction stream 14 in particular (see Fig. 3). The operation codes are transferred in parallel with other data (eg. edge crossings, fill data, etc) through the pipeline of processing stages shown in Fig. 3 and the pixel compositing module 700 places the *op* codes onto the stack in the order determined as a consequence of priority determination, level activation and fill determination.

The operation of the pixel output module 800 will now be described. Incoming messages are read from the pixel output FIFO, which include pixel output messages, repeat messages, and end of scan line messages are processed in sequence.

Upon receipt of a pixel output message the pixel output module 800 stores the pixel and also forwards the pixel to its output. Upon receipt of a repeat message the last stored pixel is forwarded to the output 898 as many times as specified by the count from the repeat message. Upon receipt of an end of scan line message the pixel output module 800 passes the message to its output.

The output 898 may connect as required to any device that utilizes pixel image data. Such devices include output devices such as video display units or printers, or memory storage devices such as hard disk, semiconductor RAM including line, band or frame stores, or a computer network.

One difficulty with implementing the system described so far is that, in many cases, it does not deal adequately with compositing of objects comprising pixels having associated opacity values. In particular, in some situations, overlapping objects are incorrectly composited at points along a scan line where one or more of the objects is not active. The situation in the present embodiment arises with Porter and Duff compositing, and will be described with reference to that particular compositing scheme. However, it

will be appreciated that the aspects of the invention described in detail below can also be applied to similar difficulties arising under other compositing schemes.

For example, the expression *PAGE ← (A in B) over (C out D) over PAGE* can be represented as a binary tree structure 1000, as shown in Fig. 22, where *A*, *B*, *C* and *D* are different graphical objects being defined by edges, a viewing priority and opacity data. A level activation table 1001 for this example is shown in Fig. 23, in which it will be seen that line 1002 implements the *C out D* portion of the expression shown in Fig. 22. Unfortunately, using this configuration, if *D* is inactive at a certain pixel, the *out* operation will incorrectly be performed with *PAGE* rather than *D*. If *C* is not opaque, this results in a visually incorrect appearance upon rendering. To avoid this type of problem, where objects having an opacity which is potentially less than 1, an inactive object must not appear in a compositing expression at a position other than to the left of an *over* operation. Given that most objects will be inactive for at least some portion of every scan line on which they appear, this can be a major problem.

One way of overcoming the problem is to "pad out" objects, using transparent pixels. In this way, the Porter and Duff expression is correctly applied, because each object effectively exists at every point at which the object it is being composited with exists. Applying this to the example above, each leaf node (ie, *A*, *B*, *C* and *D*) in the binary tree 1000 is replaced with an expression placing it *over* a transparent (glass) object the size of the bounding box around a union of the pairs of objects, as illustrated in Figs. 24 and 25.

In Fig. 24, two transparent glass bounding boxes G1 and G2 are shown. G1 represents a bounding box of *A* and *B*, whilst G2 represents a bounding box of *C* and *D*. Implementing this in the expression tree 1000 results in a modified expression tree 1004, shown in Fig. 25. A corresponding level allocation table 1005 is shown in Fig. 26. As will be apparent from the level allocation table 1005 and the modified expression tree 1004, this modification results in a substantial increase in the number of stack operations required for each pixel. This is because stack operations are performed where one or both of a pair of pixels being composited is transparent. In terms of the actual

rendered output ultimately achieved, these operations are relatively wasteful of stack resources and processor time.

One way of reducing the number of operations is to clip out the glass objects with the leaf-node objects they are composited with. The *over* operation between leaf nodes and the transparent objects is then no longer required. In view of the fact that clipping operations are, in general, less processor intensive than multiple compositing operations, this results in more efficient operation of the stack.

A level activation table 1006 for the example above using clipping of the transparent boxes G1 and G2 with *A* and *B*, and *C* and *D*, respectively, is shown in Fig. 27. It will be noted that only eight stack operations are performed per pixel in this implementation, compared with twelve operations in the case where *A*, *B*, *C* and *D* are padded using multiple *over* operations. Four of these levels are clipping levels 999. It will be appreciated, however, that a number of redundant compositing operations involving transparent pixels are still required.

A preferred method of correctly implementing Porter and Duff compositing where one or more over lapping objects are non-opaque will now be described. Referring to the example shown in Fig. 28, it will be noted that the operation *C out D* can be represented in three ways, one for each of regions $C \cap \overline{D}$, $C \cap D$ and $\overline{C} \cap D$. Region $C \cap \overline{D}$ can be represented by *C*, since there is no contribution from *D* in that region for an *out* operation, and *C* is required. In region $C \cap D$, both *C* and *D* are active, and so compositing using the *C out D* operation is required. There is no contribution from *C* or *D* in the region $\overline{C} \cap D$, and so the region need not be considered when building the compositing stack for pixels in that region.

A simplified representation of the *C out D* operation is given in Figs 42 and 43. In Fig. 42 there are shown objects *C* and *D*, for which the *out* operation is to be performed. As described above, the scan line can be divided into three regions: $C \cap \overline{D}$, $C \cap D$ and $C \cap \overline{D}$. In the present case, if *C* is non-opaque, the *out* operation will incorrectly be performed with *PAGE* in the region $C \cap \overline{D}$.

Fig. 43 illustrates the solution previously described. It will be noted that each of the objects is clipped to the region in which it is active. Region $C \cap \overline{D}$ is simply represented by a level *C1*, region $C \cap D$ is represented levels *C1* and *D1*, whilst region $\overline{C} \cap D$ requires no input from either level. It will be appreciated that using the specific requirements of particular operations to control clipping of objects to only required regions results in a considerably smaller number of compositing operations, as transparent padding pixels are no longer required.

A level activation table 1008 for implementing the clipping of this example is shown in Fig. 29. It will be seen that four additional levels are required in the level activation table 1008 compared to the previous method described. However, three out of four of these additional levels are clipping levels 1015, which are typically less intensive than compositing operations. Also, the actual compositing stack produced for each pixel is, on average, smaller than that for the earlier method described. This is due to the fact that clipping operations are used to ensure that compositing operations only take place in the regions where they are relevant. In effect, processor time is not wasted compositing pixels which do not contribute to the final image.

Compositing stack operations for non-trivial Porter and Duff compositing operators are shown in Figs 30 and 31. Table 1010 shown in Fig. 30 assumes that *D* is already on top of the stack where *D* is active, and the compositing operators are broken down according to the regions to which they apply. It will be noted that alpha channel (opacity) operation is restricted to the intersection region $S \cap D$ by activating on *S* edges and clipping with *D* edges. In the cases marked with asterisks, all of the objects which contribute to *D* need to be clipped in by the edges of *S*.

The table 1016 shown in Fig. 31 provides the operations where the *S* pixel value is already on the stack, in those regions where it is active. Note that in such a case, the *D* pixel value is on the next level down, where *D* is also active. In the cases marked with asterisks, the stack would be in an incorrect state; either the objects in the "active branch" must be clipped by the objects in the "inactive" branch to prevent the state from occurring, or the top value must be popped off the stack.

For complicated expressions, the objects may have been built up from several other objects, so that the shape of the regions *S* and *D* need to be determined from the shapes that were used to build them up. The steps required to implement this more complicated case will be apparent to those skilled in the art of Porter and Duff compositing operations.

The resulting active regions produced by the various compositing operations are given in table 1011 shown in Fig. 32.

When analysing an expression tree for rendering using the preferred embodiments, an application needs effectively to determine the active region of each branch node of the tree and pass it on to the next node up the tree. A detailed analysis of the expression tree shown in Fig. 22 will now be undertaken along these lines, in conjunction with Figs. 33 to 41.

Fig. 33 shows a preliminary analysis of the expression tree 1000 shown in Fig. 22, on the basis of active regions arising from the operations associated with each node. Referring to Fig. 34, it can be seen that the *A in B* expression is limited in activity to the intersection region of *A* and *B*. Accordingly, the edges of object *A* are used to clip object *B*, so that it does not leave colour on the stack where it is not required. Similarly, the level which performs the *in* operation is restricted to the intersection region by object *B*. The necessary stack operations to achieve this are shown in Fig. 35.

The same effect could be obtained by using the edges of object *B* to activate both levels, and clipping both with object *A*, saving a clipping level. However, both clipping objects will be required by objects higher up the expression tree.

The next branch is the *over* operation, as shown in Fig. 36, in which the active regions for the two branches are as shown. The regions to be considered are shown in Fig. 37. The boxed text 1012 at the right of Fig. 37 shows the state of the stack for each of the corresponding regions 1013 to the left.

In the latter two cases, the result of the *over* operation is already on the stack. Accordingly, the intersection region is the only one where a further operation is needed. The resulting level activation table entries are shown in Fig. 38. It will be seen that object

C activates level *O1*, and *A* and *B* clip it to the region $A \cap B \cap C$. The operation as a whole contributes to pixels in the region $(A \cap B) \cup C$.

This region is passed up to the next node in the expression tree, being the final *over*, which composites the expression onto *PAGE*. The relevant regions to be considered for each of the two branches are shown in Fig. 40, where, again, boxed text 1014 at the right hand side shows the state of the stack for the corresponding active regions 1015 on the left hand side.

To obtain correct operation of the pixel stack for a union operation requires some effort. One solution is to break up the area into mutually exclusive areas, and to use two levels for the same operation. This can be done using the active regions of each branch of the previous node to activate one of the levels. The active region of one of the levels is then used to clip out the other. For example, area *C* and area $A \cap B \cap \overline{C}$ can be used to make up $(A \cap B) \cup C$, as will be appreciated by those skilled in the art.

The resulting entries for the level activation table 1020 for the example expression are shown in Fig. 41. Levels *O2* (activated by object *C*) and *O3* (activated by object *A*, clipped in by *B* and clipped out by *C*) combine to clip the final *over* operation to the region $(A \cap B) \cup C$.

It will be noted that the clipping levels do not contribute to the compositing of the pixels. The number of contributing levels, and therefore of stack operations for each pixel, is, on average, considerably less than for the method where levels are padded out with transparent pixels. Also reduced is the number of pixels for which a particular level can be expected to be active in a given scan line.

It will be appreciated by those skilled in the art that compositing of objects over larger numbers of levels is also possible by extrapolating the method described herein. Furthermore, it will also be appreciated that the various manipulations shown can be used in different compositing paradigms, including framestore-based systems and other stack-based, line- or band-based compositing systems.

It will be apparent from the foregoing that the method and apparatus described provide for the rendering of graphic objects with full functionality demanded by

sophisticated graphic description languages without a need for intermediate storage of pixel image data during the rendering process.

The foregoing describes only a number of embodiments of the present invention, and modifications, may be made thereto without departing from the spirit and scope of the present invention, various aspects of which are appended hereto.

4. Brief Description of the Drawings

Fig. 1 is a schematic block diagram representation of a computer system incorporating the preferred embodiment;

Fig. 2 is a block diagram showing the functional data flow of the preferred embodiment;

Fig. 3 is a schematic block diagram representation of the pixel sequential rendering apparatus and associated display list and temporary stores of the preferred embodiment;

Fig. 4 is a schematic functional representation of the edge processing module of Fig. 2;

Fig. 5 is a schematic functional representation of the priority determination module of Fig. 2;

Fig. 6 is a schematic functional representation of the fill data determination module of Fig. 2;

Figs. 7A to 7C illustrate pixel combinations between source and destination;

Fig. 8 illustrates a two-object image used as an example for explaining the operation of preferred embodiment;

Figs. 9A and 9B illustrate the vector edges of the objects of Fig. 8;

Fig. 10 illustrates the rendering of a number of scan lines of the image of Fig. 8;

Fig. 11 depicts the arrangement of an edge record for the image of Fig. 8;

Figs. 12A to 12J illustrate the edge update routine implemented by the arrangement of Fig. 4 for the example of Fig. 10;

Figs. 13A and 13B illustrate the odd-even and non-zero winding fill rules;

Figs. 14A to 14E illustrate how large changes in X coordinates contribute to spill conditions and how they are handled;

Figs. 15A to 15E illustrates the priority filling routine implemented by the arrangement of Fig. 5;

Figs. 16A to 16D provide a comparison between two prior art edge description formats and that used in the preferred embodiment;

Figs. 17A and 17B show a simple compositing expression illustrated as an expression tree and a corresponding depiction;

Fig. 18A depicts a table configured for ensuring accurate stack operations;

Fig. 18B is a preferred form of the table of Fig. 18A;

Fig. 19 is an example expression tree;

Figs 20A to 20F depict an activity table evaluation of the expression of Fig. 19 and the corresponding compositing stacks during such evaluation;

Figs. 20G to 20I depict various configurations of the activity table and associated modules;

Fig. 21 depicts the result of a number of compositing operations;

Fig. 22 shows an expression tree for implementing a series of Porter and Duff compositing operations on objects A, B, C and D;

Fig 23 shows a level activation table for implementing the binary tree structure shown in Fig. 22;

Fig. 24 shows the objects of the binary tree shown in Figure 22 placed over transparent glass objects;

Fig. 25 shows a modified expression tree for implementing the arrangement shown in Fig. 24;

Fig. 26 shows a level activation table for implementing the expression tree of Figure 25;

Fig. 27 shows an alternative level activation table for the example shown in Figs. 24 and 25, in which the transparent boxes are clipped with A, B, C and D;

Fig. 28 shows an expression tree for the operation *C out D*;

Fig. 29 shows a level activation table for implementing the clipping shown in Fig. 28;

Figs. 30 and 31 show compositing stack operations for non-trivial Porter and Duff compositing operators;

Fig. 32 shows active regions for various operators;

Fig. 33 shows a similar expression tree to that shown in Fig. 22, with a preliminary analysis of the steps involved in implementing the expression;

Fig. 34 shows the active regions for the *A in B* expression from the expression tree shown in Fig. 33;

Fig. 35 shows stack operations necessary to perform the *in* operation shown in Fig. 34;

Fig. 36 is a detailed view of the *over* operation of Fig. 33, showing active regions from the *in* and *out* operations below;

Fig. 37 shows a state of the stack for each of the regions to be considered in relation to the expression of Fig. 36;

Fig. 38 shows level activation table entries for implementing the operation shown in Figs. 36 and 37;

Fig. 39 shows the final *over* operation from the expression tree of Fig. 33, indicating the active regions from the *over* and page expressions below;

Fig. 40 shows the regions to be considered in the expression of Fig. 39;

Fig. 41 shows the resulting entries for the level activation table for implementing the expression shown in Fig. 39;

Fig. 42 shows a *C out D* operation prior to implementing clipping to active regions; and

Fig. 43 shows the *C out D* operation after clipping the individual levels to their active regions.

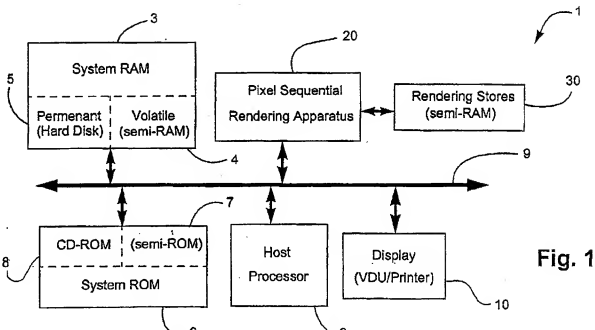


Fig. 1

(131)
特開2000-149035

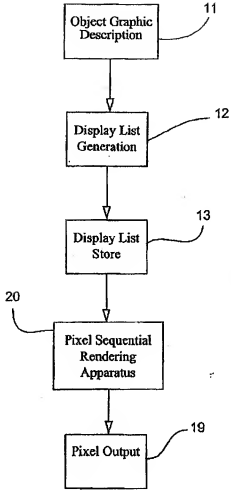

Object Graphic Description
11
Display List Generation
12
Display List Store
13
20
Pixel Sequential Rendering Apparatus
Pixel Output
19

Fig. 2

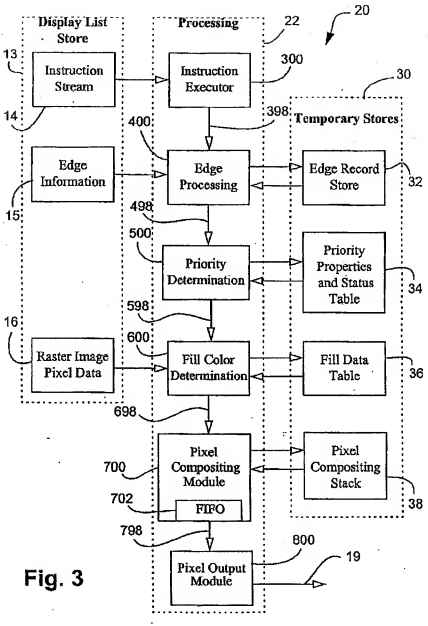

Display List Store
Processing
22
20
13
Instruction Stream
Instruction Executor
300
30
14
398
Temporary Stores
400
Edge Information
Edge Processing
Edge Record Store
32
15
498
500
Priority Determination
Priority Properties and Status Table
34
598
16
600
Raster Image Pixel Data
Fill Color Determination
Fill Data Table
36
698
Pixel Compositing Module
Pixel Compositing Stack
700
38
702
FIFO
798
800
19
Pixel Output Module

Fig. 3

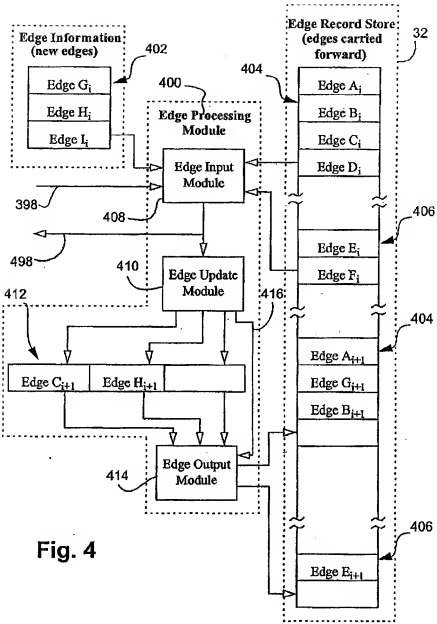

Edge Information (new edges)
402
Edge G_i
Edge H_i
Edge I_i
400
Edge Processing Module
Edge Input Module
398
408
498
410
Edge Update Module
412
416
Edge C_{i+1}
Edge H_{i+1}
414
Edge Output Module
Edge Record Store (edges carried forward)
32
404
Edge A_i
Edge B_i
Edge C_i
Edge D_i
406
Edge E_i
Edge F_i
404
Edge A_{i+1}
Edge G_{i+1}
Edge B_{i+1}
406
Edge E_{i+1}

Fig. 4

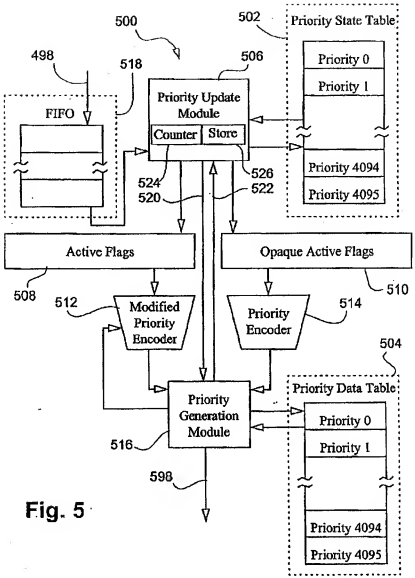

500
502
Priority State Table
498
518
506
Priority 0
Priority 1
FIFO
Priority Update Module
Counter
Store
Priority 4094
Priority 4095
526
524
522
520
Active Flags
Opaque Active Flags
508
512
510
Modified Priority Encoder
Priority Encoder
514
504
Priority Data Table
Priority Generation Module
Priority 0
Priority 1
516
598
Priority 4094
Priority 4095

Fig. 5

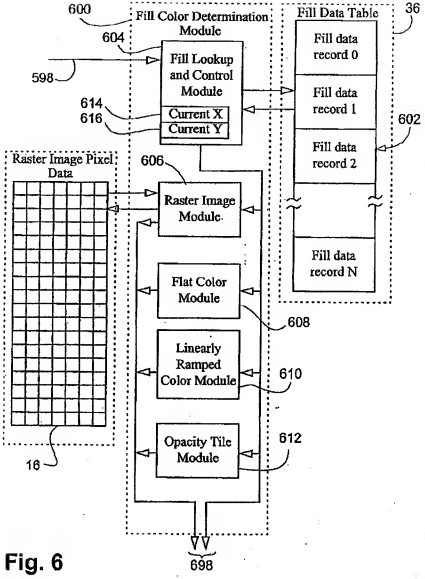

600
Fill Color Determination Module
604
Fill Lookup and Control Module
598
614
Current X
616
Current Y
Fill Data Table
36
Fill data record 0
Fill data record 1
Fill data record 2
602
Fill data record N
Raster Image Pixel Data
606
Raster Image Module
Flat Color Module
608
Linearly Ramped Color Module
610
Opacity Tile Module
612
16
698

Fig. 6

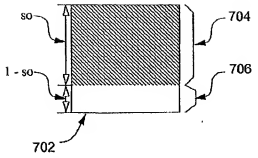


Fig. 7A

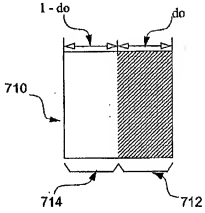


Fig. 7B

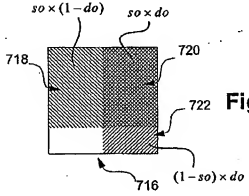


Fig. 7C

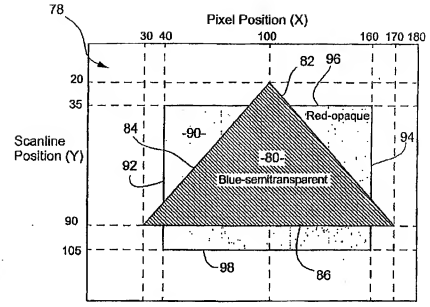


Fig. 8

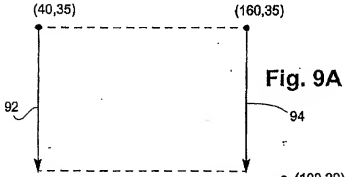


Fig. 9A

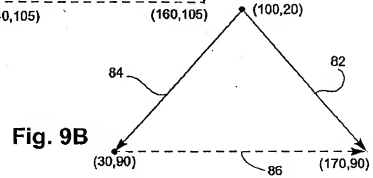


Fig. 9B

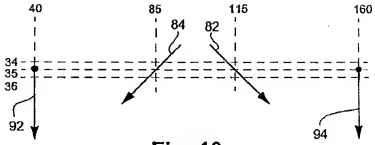

40
85
84
82
115
160
34
35
36
92
94

Fig. 10

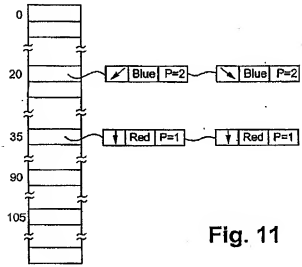

0
20
Blue P=2
Blue P=2
35
Red P=1
Red P=1
90
105

Fig. 11

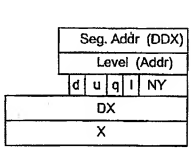


Fig. 12A

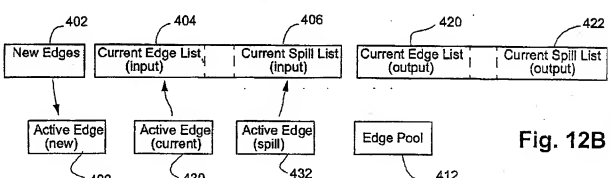


Fig. 12B

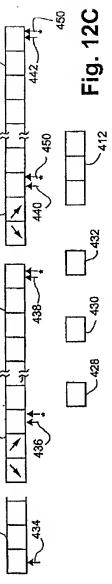


Fig. 12C

Fig. 12D

Fig. 12E

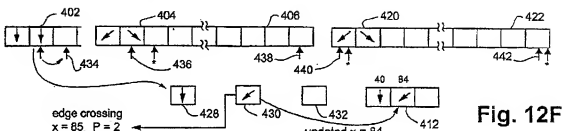


Fig. 12F

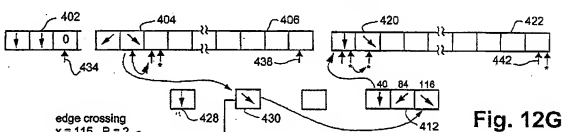


Fig. 12G

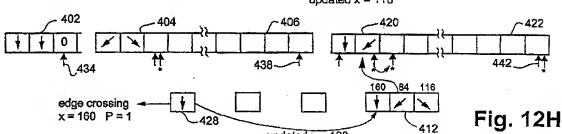


Fig. 12H

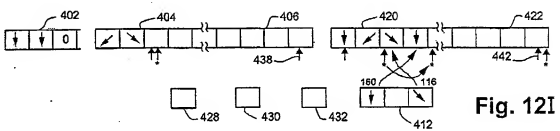


Fig. 12I

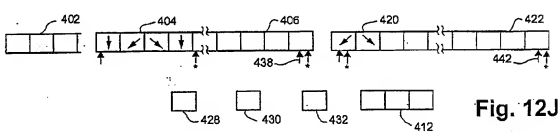


Fig. 12J

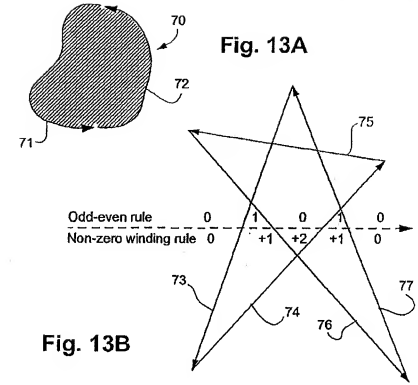


Fig. 13A

Fig. 13B

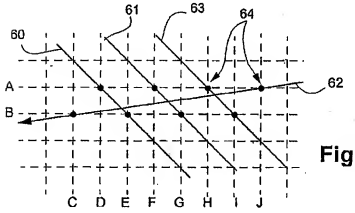


Fig. 14A

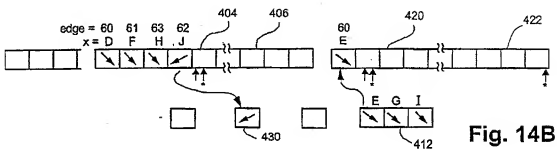


Fig. 14B

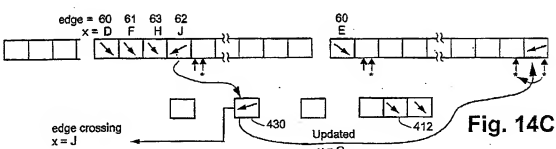


Fig. 14C

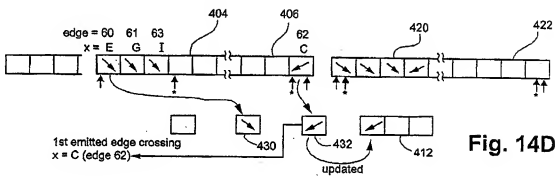


Fig. 14D

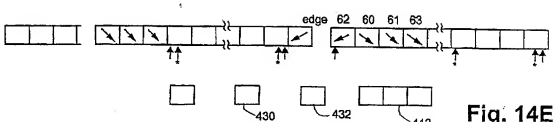


Fig. 14E

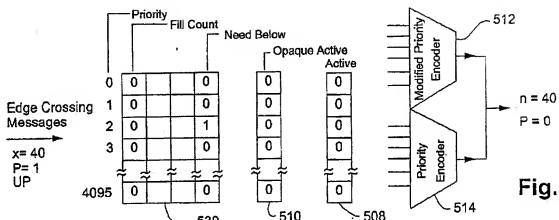


Fig. 15A

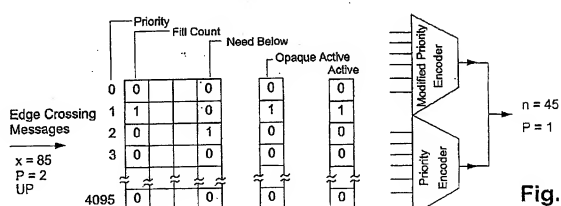


Fig. 15B

特願2000-149035

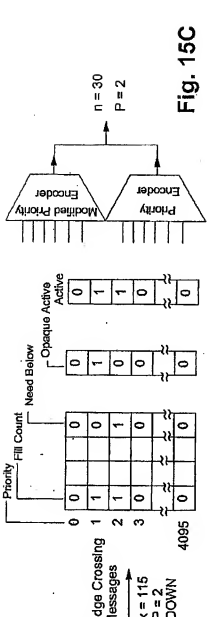


Fig. 15C

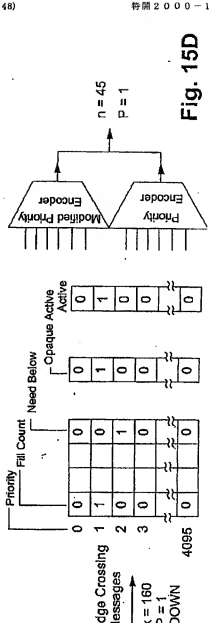


Fig. 15D

New Line
Priority
Fill Count
Need Below
Opaque Active
Active
0
1
2
3
4095
Modified Priority Encoder
Priority Encoder
n = 20
P = 0

Fig. 15E

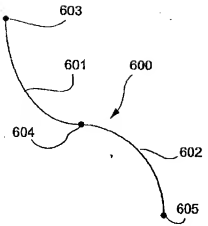


Fig. 16A
(Prior Art)

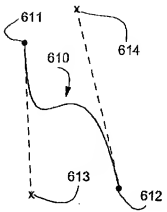


Fig. 16B
(Prior Art)

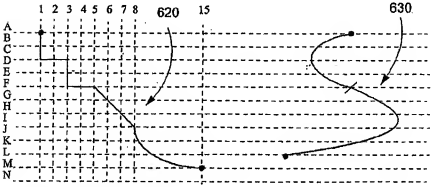


Fig. 16C

Fig. 16D

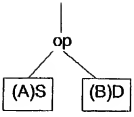


Fig. 17A

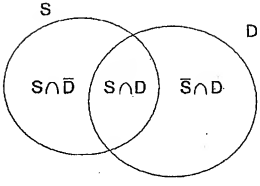


Fig. 17B

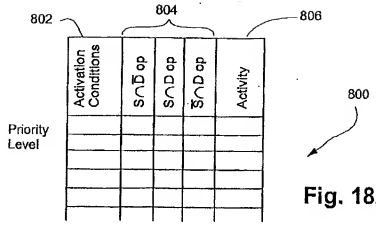


Fig. 18A

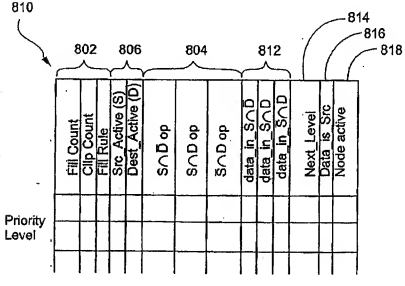
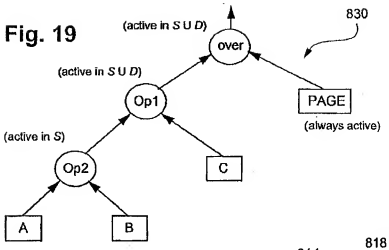


Fig. 18B

Fig. 19



	Fill Count	Clip Count	Fill Rule	Src_Active (S)	Dest_Active (D)	S∩D op	S∩D op	S∩D op	data_in_S∩D	data_in_S∩D	data_in_S∩D	Next Level	Data_Is_Src (Data is Dest)	Node active
						...	...	...						
over				1	1	NOP	over	NOP	1	1	1		?	1
Op1				1	0	Op1a	Op1b	Op1c	1	1	1	over*	1	1
AOp2				1	0	AOp2aB	AOp2bB	PopB	1	1	0	Op1*	1	1
B					0	Push B	Push B	NOP	0	0	0	AOp2*	0	0
C					0	Push C	Push C	NOP	0	0	0	Op1*	0	0

802 806 804 814 812 816 818 820

Fig. 20A

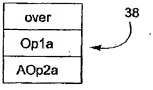


Fig. 20B

Fig. 20D

	Fill Count	Clip Count	Fill Rule	Src_Active (S)	Dest_Active (D)	S∩D op	S∩D op	S∩D op	data_in S∩D	data_in S∩D	data_in S∩D	Next Level	Data_is_Src (Data is Dest)	Node active
...						...	...	...						
over				1	1	NOP	over	NOP	1	1	1		?	1
Op1				1	0	Op1a	Op1b	Op1c	1	1	1	over*	1	1
AOp2				1	1	AOp2aB	AOp2bB	PopB	1	1	0	Op1*	1	1
B					1	Push B	Push B	NOP	1	1	0	AOp2*	0	1
C					0	Push C	Push C	NOP	0	0	0	Op1*	0	0

802 806 804 814 812 816 818 820

Fig. 20C

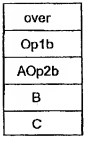


Fig. 20F

	Fill Count	Clip Count	Fill Rule	Src_Active (S)	Dest_Active (D)	S∩D op	S∩D op	S∩D op	data_in_S∩D	data_in_S∩D	data_in_S∩D	Next Level	Data_is_Src (Data is Dest)	Node active
...						...	...	...						
over				1	1	NOP	over	NOP	1	1	1		?	1
Op1				1	1	Op1a	Op1b	Op1c	1	1	1	over*	1	1
AOp2				1	1	AOp2aB	AOp2bB	PopB	1	1	0	Op1*	1	1
B					1	Push B	Push B	NOP	1	1	0	AOp2*	0	1
C					1	Push C	Push C	NOP	1	1	0	Op1*	0	1

802 806 804 812 814 816 818 820

Fig. 20E

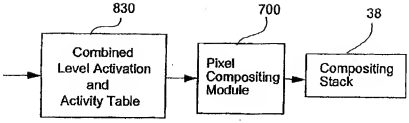


Fig. 20G

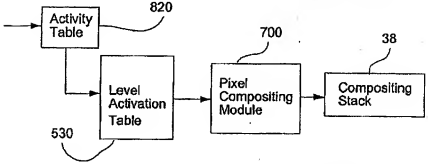


Fig. 20H

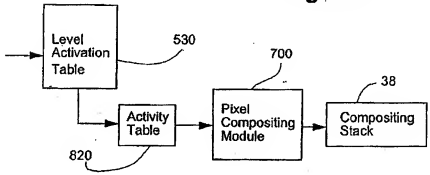


Fig. 20I

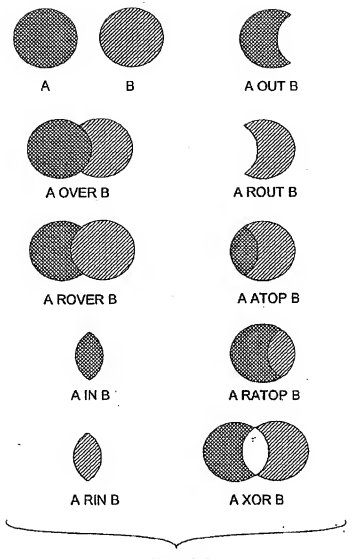

A
B
A OUT B
A OVER B
A ROUT B
A ROVER B
A ATOP B
A IN B
A RATOP B
A RIN B
A XOR B

Fig. 21

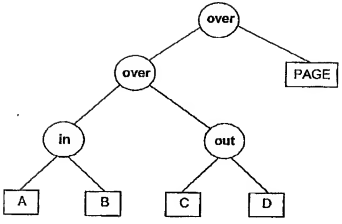


Fig. 22

Fill Type	LEVEL_CLIPPER	LEVEL_CLIP_OUT	LEVEL_NEED_BELOW	LEVEL_X_INDEPENDENT	LEVEL_STACK_OP	LAO_USE_D_OUT_S	LAO_USE_S_OUT_D	LAO_USE_S_ROP_D	LEVEL_COLOR_OP	LEVEL_ODD_EVEN	LEVEL_ATTRIBUTES	Fill Index	
Pop src ((A in B) over (C out D)) over dest (PAGE)	0	0	1		10	1	1	1	LCO_COPYPEN	1			
Pop src (A in B) over dest (C out D)		0	0	1		10	1	1	LCO_COPYPEN	1			
A in dest (B)	01	0	0	1	0	00	0	0	1	LCO_COPYPEN	1	000	A
Push B onto stack	11	0	0	1	0	01	0	1	1	LCO_COPYPEN	1	000	B
C out dest (D)	11	0	0	1	0	00	0	1	0	LCO_BLACK	1	000	C
Push D onto stack	00	0	0	1	1	01	0	1	1	LCO_COPYPEN	1	000	D
									PAGE				
									...				
									...				

Fig. 23

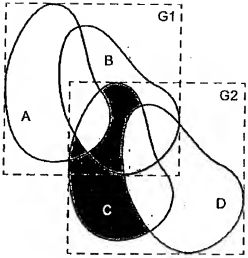

G1
B
G2
A
C
D

Fig. 24

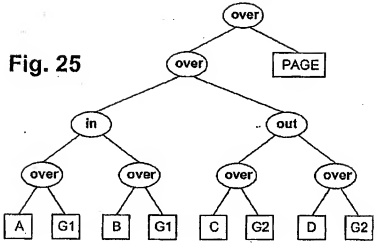

over
over
PAGE
in
out
over
over
over
over
A
G1
B
G1
C
G2
D
G2

Fig. 25

	Fill Type	LEVEL_CLIPPER	LEVEL_CLIP_OUT	LEVEL_NEED_BELOW	LEVEL_X_INDEPENDENT	LEVEL_STACK_OP	LAO_USE_D_OUT_S	LAO_USE_S_OUT_D	LAO_USE_S_ROP_D	LEVEL_COLOR_OP	LEVEL_ODD_EVEN	LEVEL_ATTRIBUTES	Fill Index
Pop src ((A In B) over (C out D)) over dest (PAGE)	0	0	1		10	1	1	1	LOG_COPYPEN	1		
Pop src (A In B) over dest (C out D)	0	0	1		10	1	1	1	LOG_COPYPEN	1		
Pop scr (A) in dest (B)	0	0	1	1	10	0	0	1	LOG_COPYPEN	1	000	in
A over G2	01	0	0	1	0	00	1	1	LOG_COPYPEN	1	000	A
	00	0	0	0	1	01	0	0	LOG_BLACK	1	000	G2
B over G2	11	0	0	1	0	00	1	1	LOG_COPYPEN	1	000	B
	00	0	0	0	1	01	0	0	LOG_BLACK	1	000	G2
Pop scr (C) out dest (D)	0	0	1	0	1	10	0	1	LOG_BLACK	1	000	out
C over G1	11	0	0	1	0	00	1	1	LOG_COPYPEN	1	000	C
	00	0	0	1	1	01	0	0	LOG_BLACK	1	000	G1
D over G1	00	0	0	1	1	00	1	1	LOG_COPYPEN	1	000	D
	00	0	0	1	1	01	0	0	LOG_BLACK	1	000	G1
									PAGE			

Fig. 26

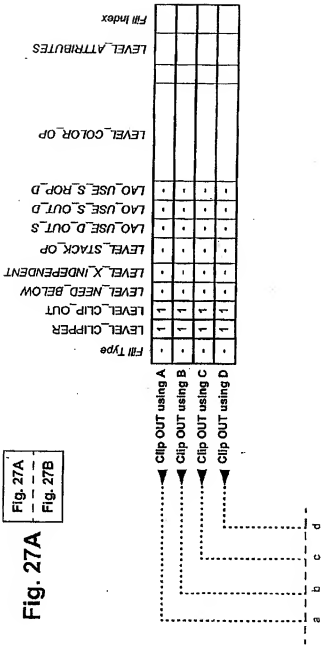


Fig. 27A

Fig. 27A
Fig. 27B

Fig. 27B

Fig. 27A
Fig. 27B

		Fill Type	LEVEL_CLIPPER	LEVEL_CLIP_OUT	LEVEL_NEED_BELOW	LEVEL_X_INDEPENDENT	LEVEL_STACK_OP	LAO_USE_D_OUT_S	LAO_USE_S_OUT_D	LAO_USE_S_ROP_D	LEVEL_COLOR_OP	LEVEL_ODD_EVEN	LEVEL_ATTRIBUTES	Fill Index
a	Pop src'((A in B) over (C out D)) over dest (PAGE)	0	0	1		10	1	1	1	LCO_COPYPEN	1		
b	Pop src (A in B) over dest (C out D)		0	1		10	1	1	1	LCO_COPYPEN	1		
c	Pop scr (A) in dest (B)			0	0	1	0	0	1	LCO_COPYPEN	1	000	In
d		Push A	01	0	0	1	0	1	1	LCO_COPYPEN	1	000	A
		G2	00	0	0	1	1	01	0	LCO_BLACK	1	000	-
		Push B	11	0	0	1	0	1	1	LCO_COPYPEN	1	000	B
		G2	00	0	0	1	1	01	0	LCO_BLACK	1	000	-
	Pop scr (C) out dest (D)			0	0	1	0	10	0	LCO_BLACK	1	000	out
		Push C	11	0	0	1	0	01	0	LCO_COPYPEN	1	000	C
		G1	00	0	0	1	1	01	0	LCO_BLACK	1	000	-
		Push D	00	0	0	1	1	01	0	LCO_COPYPEN	1	000	D
		G1	00	0	1	1	0	01	0	LCO_BLACK	1	000	-
										PAGE				

In region C ∩ D̄
In region C ∩ D
In region C̄ ∩ D
C
out
D
C
(Inactive)
C ∩ D̄
C ∩ D
C̄ ∩ D

Fig. 28

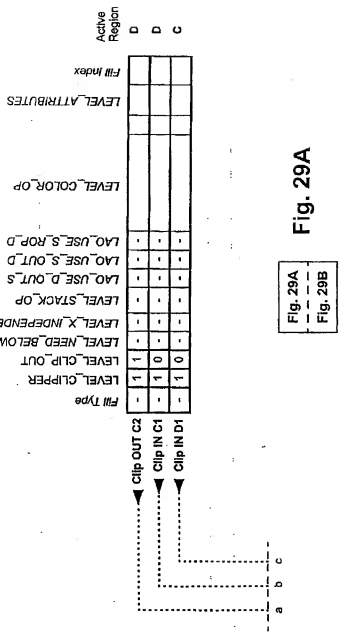


Fig. 29A

Fig. 29A
Fig. 29B

Fig. 29A
Fig. 29B

Fig. 29B

Operator	Region	Compositing Stack Operations					Fill Object
		LEVEL_ COLOR_OP	S_ROP_D	S_OUT_D	D_OUT_S		
S over D	$\overline{S \cap D}$	Result is on the Stack					
	$S \cap D$	LCO_COPYPEN	1	1	1		S
	$S \cap \overline{D}$	LCO_COPYPEN	1	1	0		S
S rover D	$\overline{S \cap D}$	Result is on the Stack					
	$S \cap D$	LCO_NOP	1	1	1		S
	$S \cap \overline{D}$	LCO_COPYPEN	1	1	0		S
S in D	$\overline{S \cap D}$	* Clip D with edges of S					
	$S \cap D$	LCO_COPYPEN	1	0	0		S
	$S \cap \overline{D}$	No Operation					
S rin D	$\overline{S \cap D}$	* Clip D with edges of S					
	$S \cap D$	LCO_NOP	1	0	0		S
	$S \cap \overline{D}$	No Operation					
S out D	$\overline{S \cap D}$	* Clip D with edges of S					
	$S \cap D$	any	0	1	0		S
	$S \cap \overline{D}$	LCO_COPYPEN	1	1	0		S
S rout D	$\overline{S \cap D}$	Result is on the Stack					
	$S \cap D$	any	0	0	1		S
	$S \cap \overline{D}$	No Operation					
S atop D	$\overline{S \cap D}$	Result is on the Stack					
	$S \cap D$	LCO_COPYPEN	1	0	1		S
	$S \cap \overline{D}$	No Operation					
S ratop D	$\overline{S \cap D}$	* Clip D with edges of S					
	$S \cap D$	LCO_NOP	1	1	0		S
	$S \cap \overline{D}$	LCO_COPYPEN	1	1	0		S
S xor D	$\overline{S \cap D}$	Result is on the Stack					
	$S \cap D$	any	0	1	1		S
	$S \cap \overline{D}$	LCO_COPYPEN	1	1	0		S

Fig. 30

Operator	Region	Compositing Stack Operations				
		STACK_OP	LEVEL_ COLOR_OP	S_ROP_D	S_OUT_D	D_OUT_S
S over D	$\overline{S \cap D}$	Result on Stack				
	$S \cap D$	10	LCO_COPYPEN	1	1	1
	$S \cap \overline{D}$	Result on stack				
S rover D	$\overline{S \cap D}$	Result on stack				
	$S \cap D$	10	LCO_NOP	1	1	1
	$S \cap \overline{D}$	Result on Stack				
S in D	$\overline{S \cap D}$	* Clip D with S edges				
	$S \cap D$	10	LCO_COPYPEN	1	0	0
	$S \cap \overline{D}$	* Clip S with D edges				
S rin D	$\overline{S \cap D}$	* Clip D with S edges				
	$S \cap D$	10	LCO_NOP	1	0	0
	$S \cap \overline{D}$	* Clip S with D edges				
S out D	$\overline{S \cap D}$	* Clip D with S edges				
	$S \cap D$	10	any	0	1	0
	$S \cap \overline{D}$	Result on stack				
S rout D	$\overline{S \cap D}$	Result on stack				
	$S \cap D$	10	any	0	0	1
	$S \cap \overline{D}$	* Clip S with D edges				
S atop D	$\overline{S \cap D}$	Result on stack				
	$S \cap D$	10	LCO_COPYPEN	1	0	1
	$S \cap \overline{D}$	* Clip S with D edges				
S ratop D	$\overline{S \cap D}$	* Clip D with S edges				
	$S \cap D$	10	LCO_NOP	1	1	0
	$S \cap \overline{D}$	Result on stack				
S xor D	$\overline{S \cap D}$	Result on stack				
	$S \cap D$	10	any	0	1	1
	$S \cap \overline{D}$	Result on stack				

Fig. 31

Fig. 32

Operator	Active Region
S over D	$S \cup D$
S rover D	$S \cup D$
S in D	$S \cap D$
S rin D	$S \cap D$
S out D	S
S rout D	D
S atop D	D
S ratop D	S
S xor D	$S \cup D$

Fig. 33

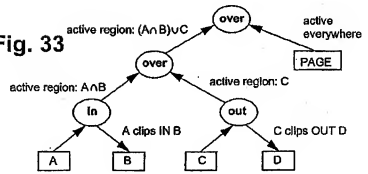
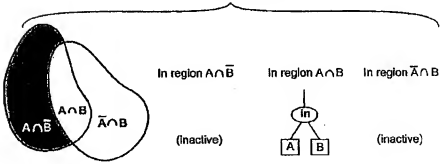


Fig. 34



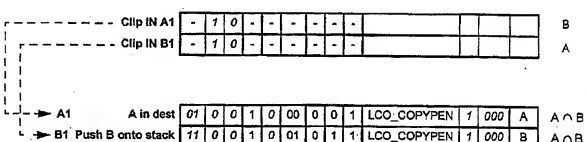

Clip IN A1
Clip IN B1
A1
A in dest
B1
Push B onto stack
LCO_COPYPEN
LCO_COPYPEN
B
A
A∩B
A∩B

Fig. 35

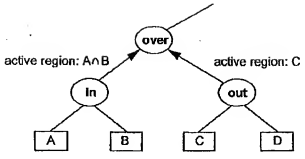

over
active region: A∩B
active region: C
in
out
A
B
C
D

Fig. 36

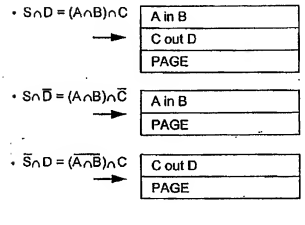

• S∩D = (A∩B)∩C
A in B
C out D
PAGE
• S∩D̄ = (A∩B)∩C̄
A in B
PAGE
• S̄∩D = (A∩B̄)∩C
C out D
PAGE

Fig. 37

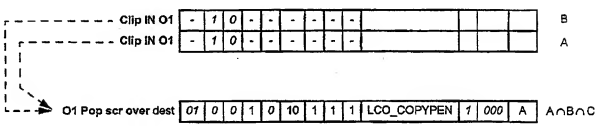


Fig. 38

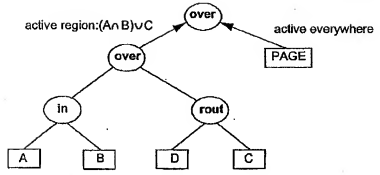

active region:(A∩B)∪C
over
active everywhere
PAGE
over
in
rout
A
B
D
C

Fig. 39

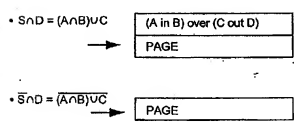

• S∩D = (A∩B)∪C
(A in B) over (C out D)
PAGE
• $\overline{S}$∩D = $\overline{(A \cap B) \cup C}$
PAGE

Fig. 40

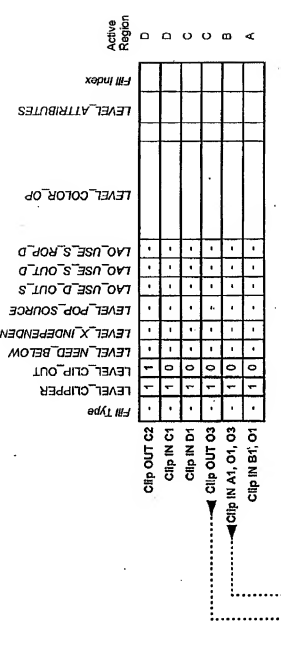


Fig. 41A

Fig. 41A
Fig. 41B

Fig. 41A
Fig. 41B

Fig. 41B

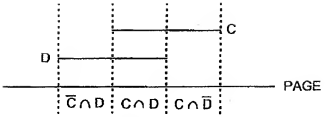


Fig. 42

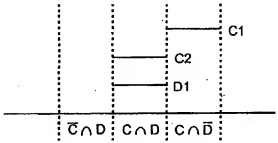


Fig. 43

1. ABSTRACT

Disclosed are methods, apparatus (1) and computer readable media for rendering at least one graphic object (80, 90) described by at least one edge (82-86, 92-98) into a raster pixel image (78) having a plurality of scan lines and a plurality of pixel locations on each scan line. For each scan line, coordinates of intersection of those edges of the objects that intersect the scan line are determined in a predetermined order. This is preferably achieved by processing edge records (418) using a number of buffers (402, 404,406, 412,420, 422) thereby enabling efficient sorting of edge intersections into order. For each adjacent pair of edge intersections, information (530) associated with the corresponding object is examined to determining a set of active objects (508, 510) for a span of pixel locations between the corresponding pair of edge intersections. For each span of pixel locations, the corresponding set of active objects is used to determine (600) a value for each of the locations within the span. The information may include one or more of a fill count, a clip count and other factors. A compositing model accommodating opacity is also disclosed, as are stack operations used to facilitate rendering and other features which contribute to fast processing of image components.

2. Representative Drawing

Fig. 1